**Java QueryDBOracle-Api**

# API Rest Java - consultas DB Oracle

Crear API Rest mediante unos de Spring – Java como Backend.

* Crear controladores

@RestController

@CrossOrigin(origins = "\*", maxAge = 3600)

@RequestMapping("/api")

**public** **class** ControllerOracle {

**private** **final** ServiceOracle serviceOracle;

**private** **final** ServicePassword servicePassword;

/\*\*

\* Inyecta automaticamente la clase ServiceOracle

\*

\* **@param** service

\*/

@Autowired

**public** ControllerOracle(ServiceOracle serviceOracle, ServicePassword servicePassword) {

**this**.serviceOracle = serviceOracle;

**this**.servicePassword = servicePassword;

}

…

@RestController: Expone servicios GET, POST, PUT y DELETE para publicar servicios API Rest

@CrossOrigin(origins = "\*", maxAge = 3600): Permite conexiones de cualquier IP Cross CORS

@RequestMapping("/api"): Agregar /api a la ruta de los servicios expuestos

* Crear Servicios
* Consultar DB

package com.wlopera.query.oracle.service;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.google.gson.Gson;

import com.google.gson.JsonElement;

import com.google.gson.JsonParser;

import com.wlopera.query.oracle.DatabaseConfig;

import com.wlopera.query.oracle.connection.Connect;

import com.wlopera.query.oracle.mapper.Mapper;

import com.wlopera.query.oracle.utilities.Utility;

@Service

public class ServiceOracle {

private final DatabaseConfig dbConfig;

/\*\*

\* Inyecta automaticamente el objeto databaseConfig

\*

\* @param databaseConfig

\*/

@Autowired

public ServiceOracle(DatabaseConfig dbConfig) {

this.dbConfig = dbConfig;

}

public String getDataByQuery(String query) throws Exception {

Connection connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

connection = Connect.getConnection(dbConfig);

statement = connection.createStatement();

resultSet = statement.executeQuery(query);

List<Map<String, Object>> rows = getData(resultSet);

return Mapper.genericJsonByObject(rows);

} catch (Exception e) {

System.out.println("##=> Error consultando la tabla en oracle ");

throw e;

} finally {

if (resultSet != null && resultSet.isClosed()) {

resultSet.close();

}

if (statement != null && statement.isClosed()) {

statement.close();

}

if (connection != null && connection.isClosed()) {

connection.close();

}

}

}

public String getDataByQueryWithParams(String query, DatabaseConfig config) throws Exception {

Connection connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

connection = Connect.getConnection(config);

statement = connection.createStatement();

resultSet = statement.executeQuery(query);

List<Map<String, Object>> rows = getData(resultSet);

return Mapper.genericJsonByObject(rows);

} catch (Exception e) {

System.out.println("##=> Error consultando la tabla en oracle ");

throw e;

} finally {

if (resultSet != null && resultSet.isClosed()) {

resultSet.close();

}

if (statement != null && statement.isClosed()) {

statement.close();

}

if (connection != null && connection.isClosed()) {

connection.close();

}

}

}

/\*\*

\* Recorre la lista de resultados del query y retorna una lista de datos

\*

\* @param resultSet

\* @return

\* @throws Exception

\*/

private List<Map<String, Object>> getData(ResultSet resultSet) throws Exception {

int columnCount = resultSet.getMetaData().getColumnCount();

String column = "";

Object value = "";

List<Map<String, Object>> rows = new ArrayList<>();

Gson gson = new Gson();

while (resultSet.next()) {

Map<String, Object> row = new HashMap<>();

for (int i = 1; i <= columnCount; i++) {

column = resultSet.getMetaData().getColumnName(i);

value = resultSet.getString(resultSet.getMetaData().getColumnName(i));

if (value instanceof String && Utility.isJson((String) value)) {

try {

JsonElement jsonElement = JsonParser.parseString((String) value);

value = gson.fromJson(jsonElement, Object.class);

} catch (Exception e) {

e.printStackTrace();

}

}

row.put(column, value);

}

rows.add(row);

}

return rows;

}

}

@Service: Genera un servicio. Para consultar datos de BD se puede pasar los parámetros de conexión o tomar de un .yml por defecto.

* Encriptar o desencriptar palabra

package com.wlopera.query.oracle.service;

import javax.crypto.Cipher;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

import org.springframework.stereotype.Service;

import java.util.Base64;

@Service

public class ServicePassword {

private final String ALGORITHM = "AES";

private final String TRANSFORMATION = "AES/CBC/PKCS5Padding";

private final String iv = "0123456789ABCDEF";

private final String secretKey = "ABCDE67890QWERT6";

public String encrypt(String value) throws Exception {

if (secretKey.length() != 16) {

throw new IllegalArgumentException("La longitud de la clave debe ser de 16 bytes (128 bits)");

}

Cipher cipher = Cipher.getInstance(TRANSFORMATION);

SecretKeySpec keySpec = new SecretKeySpec(secretKey.getBytes(), ALGORITHM);

IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

cipher.init(Cipher.ENCRYPT\_MODE, keySpec, ivSpec);

byte[] encryptedBytes = cipher.doFinal(value.getBytes());

byte[] ivBytes = cipher.getIV();

byte[] result = new byte[ivBytes.length + encryptedBytes.length];

System.arraycopy(ivBytes, 0, result, 0, ivBytes.length);

System.arraycopy(encryptedBytes, 0, result, ivBytes.length, encryptedBytes.length);

return Base64.getEncoder().encodeToString(result);

}

public String decrypt(String encryptedValue) throws Exception {

if (secretKey.length() != 16) {

throw new IllegalArgumentException("La longitud de la clave debe ser de 16 bytes (128 bits)");

}

byte[] encryptedBytes = Base64.getDecoder().decode(encryptedValue);

IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

SecretKeySpec keySpec = new SecretKeySpec(secretKey.getBytes(), ALGORITHM);

Cipher cipher = Cipher.getInstance(TRANSFORMATION);

cipher.init(Cipher.DECRYPT\_MODE, keySpec, ivSpec);

int ivSize = cipher.getBlockSize();

byte[] ivBytes = new byte[ivSize];

byte[] encryptedData = new byte[encryptedBytes.length - ivSize];

System.arraycopy(encryptedBytes, 0, ivBytes, 0, ivSize);

System.arraycopy(encryptedBytes, ivSize, encryptedData, 0, encryptedData.length);

byte[] decryptedBytes = cipher.doFinal(encryptedData);

return new String(decryptedBytes);

}

public void run() throws Exception {

// Ejemplo de uso

String originalText = "william";

// Encriptar

String encryptedText = encrypt(originalText);

System.out.println("Texto cifrado: " + encryptedText);

// Desencriptar

String decryptedText = decrypt(encryptedText);

System.out.println("Texto descifrado: " + decryptedText);

}

public static void main(String[] args) throws Exception {

new ServicePassword().run();

}

}

* Crear Clases para configuración de archivos de DB

@Configuration

@ConfigurationProperties(prefix = "oracle")

**public** **class** DatabaseConfig {

…

@Configuration: Archviso de configuración

@ConfigurationProperties(prefix = "oracle"): Extrae los datos de un .yml y del tag: Oracle

Application.yml:

server.port: 8585

oracle:

host: 192.168.251.96

port: 1521

dbName: PROLINUX

user: SRV\_OLEGPROBD

password: SRV\_OLEGPROBD

Crear application – service:

package com.wlopera.query.oracle;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OracleQueryApplication {

public static void main(String[] args) {

SpringApplication.run(OracleQueryApplication.class, args);

}

}

@SpringBootApplication: Spring levanta una APP Service

* Crear Utilitarios:

Encriptador / Desencriptador:

package com.wlopera.query.oracle.utilities;

import javax.crypto.Cipher;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

import java.util.Base64;

public class StringEncrypt {

private static final String ALGORITHM = "AES";

private static final String TRANSFORMATION = "AES/CBC/PKCS5Padding";

public static String encrypt(String value, String secretKey, String iv) {

try {

if (secretKey.length() != 16) {

throw new IllegalArgumentException("La longitud de la clave debe ser de 16 bytes (128 bits)");

}

Cipher cipher = Cipher.getInstance(TRANSFORMATION);

SecretKeySpec keySpec = new SecretKeySpec(secretKey.getBytes(), ALGORITHM);

IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

cipher.init(Cipher.ENCRYPT\_MODE, keySpec, ivSpec);

byte[] encryptedBytes = cipher.doFinal(value.getBytes());

byte[] ivBytes = cipher.getIV();

byte[] result = new byte[ivBytes.length + encryptedBytes.length];

System.arraycopy(ivBytes, 0, result, 0, ivBytes.length);

System.arraycopy(encryptedBytes, 0, result, ivBytes.length, encryptedBytes.length);

return Base64.getEncoder().encodeToString(result);

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

public static String decrypt(String encryptedValue, String secretKey, String iv) {

try {

if (secretKey.length() != 16) {

throw new IllegalArgumentException("La longitud de la clave debe ser de 16 bytes (128 bits)");

}

byte[] encryptedBytes = Base64.getDecoder().decode(encryptedValue);

IvParameterSpec ivSpec = new IvParameterSpec(iv.getBytes());

SecretKeySpec keySpec = new SecretKeySpec(secretKey.getBytes(), ALGORITHM);

Cipher cipher = Cipher.getInstance(TRANSFORMATION);

cipher.init(Cipher.DECRYPT\_MODE, keySpec, ivSpec);

int ivSize = cipher.getBlockSize();

byte[] ivBytes = new byte[ivSize];

byte[] encryptedData = new byte[encryptedBytes.length - ivSize];

System.arraycopy(encryptedBytes, 0, ivBytes, 0, ivSize);

System.arraycopy(encryptedBytes, ivSize, encryptedData, 0, encryptedData.length);

byte[] decryptedBytes = cipher.doFinal(encryptedData);

return new String(decryptedBytes);

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

public static void main(String[] args) {

// Ejemplo de uso

String secretKey = "1234567890123456"; // Clave de 16 bytes (128 bits)

String iv = "0123456789ABCDEF";

String originalText = "william";

// Encriptar

String encryptedText = encrypt(originalText, secretKey, iv);

System.out.println("Texto cifrado: " + encryptedText);

// Desencriptar

String decryptedText = decrypt(encryptedText, secretKey, iv);

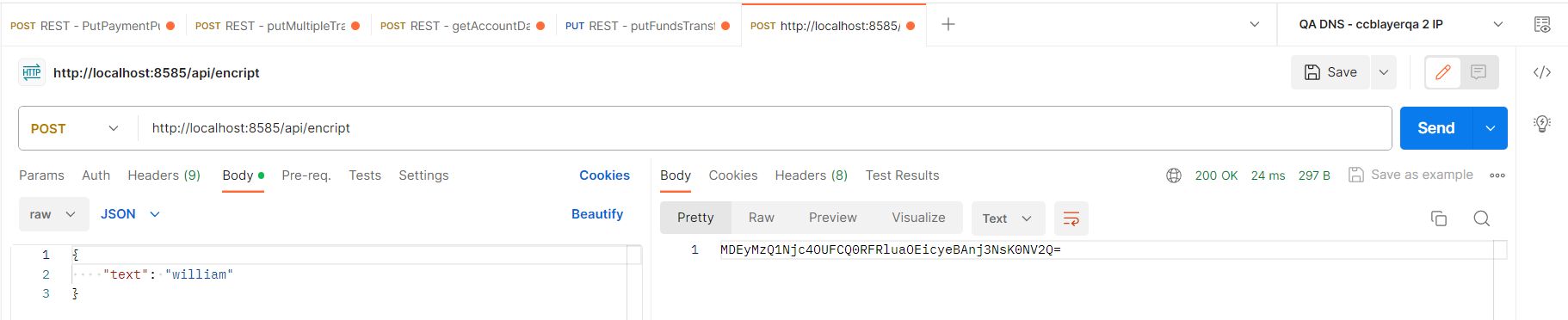
System.out.println("Texto descifrado: " + decryptedText);

}

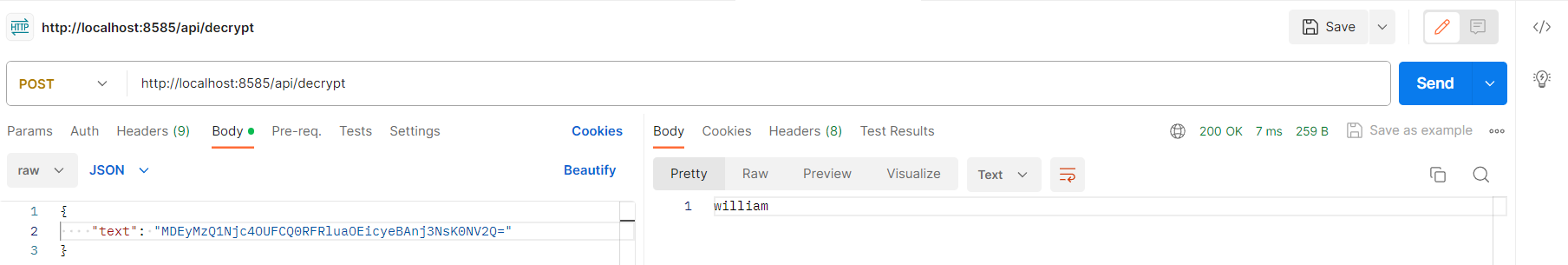
}

**Salida**

* Encriptar una palabra



* Desencriptar una palabra



* Consultar DB con o sin pasar parámetros de conexión a la base de datos. Ver proyecto de React: <https://github.com/wlopera/r-db-oracle>