**Universidad San Carlos de Guatemala**
**Facultad de Ingeniería**
**Escuela de Ciencias y Sistemas**
**Base de Datos 1, Sección N**
**Catedrático: Álvaro Longo**
**Auxiliar: Luis Ordoñez**

# MANUAL TÉCNICO

**Josué David Zea Herrera - 201807159**
**José Abraham Solorzano Herrera - 201800937**

Se creo una clase env.js que tendrá las configuraciones para la base de datos, como viene siendo el puerto, que viene siendo el localhost; host, que se usara en el 5000; userDb, que viene siendo el usuario de oracle; pass, que viene siendo la contraseña de oracle; y por último la conección, que en este caso se llama ‘localhost:1521/ORCLCDB.localdomain’.

```
module.exports = {
    NODE_ENV: process.env.NODE_ENV  || 'development',
    HOST: process.env.HOST          || 'localhost',
    PORT: process.env.PORT          || 5000,
    USERDB: process.env.USERDB      || 'useroracle',
    PASS: process.env.PASS          || 'password',
    CONN: process.env.CONN          || 'localhost:1521/ORCLCDB.localdomain'

}
```

Se creo una clase Config.js dónde se encuentra las configuraciones necesarias para conectar a la base de datos.

```
/*                  Oracle
    ------------------------------------------
    -   ¿Qué Hace el Commit adentro de la DB?   -
    - Confirmar los cambios en la base de datos -
    -   como viene siendo un insert en la DB    -
    ------------------------------------------
*/
const config = require('./env.js');
const oracledb = require('oracledb');
options = { outFormat: oracledb.OUT_FORMAT_OBJECT };
db = {
    user: config.USERDB,
    password: config.PASS,
    connectString: config.CONN
}

async function open(query, binds, autoCommit){
    var con = await oracledb.getConnection(db);
    var result = await con.execute(query, binds, {autoCommit});
    con.release();
    return result;
}

exports.Open = open;
```

Options es una variable que es la encargada de obtener, el usuario, contraseña y la conexión.

Luego se crea una función asíncrona para poner conectarnos a la base de datos por medio de al función getConnection donde se manda la variable options, luego por el comando execute se manda el query a la base de datos.

En el index.js se crea todas las importaciones necesarias, para poder acceder a la base de datos con su respectiva configuración.

```
const config = require('./config/env.js');
var bodyParser = require('body-parser');
const express = require('express');
const morgan = require('morgan');
const cors = require('cors');
const app = express();

// imports
const getRoute = require('./routes/Routes.js')

// settings
app.set('port', config.PORT);

// Middleware -- Intercambio de información entre aplicaciones.
app.use(morgan('dev'));                         // Para desarrollador.
app.use(express.json({limit: '50mb'}));                     // Las peticiones las va a devolver en formato json.
app.use(express.urlencoded({extended:false, limit: '50mb'}));  // Sirve para enviar fotos o videos, en este caso utilizaremos texto por eso es falso.
app.use(bodyParser.json({limit: '50mb'}));
app.use(bodyParser.urlencoded({limit: '50mb', extended: true}));
app.use(cors());

// Routes
app.use(getRoute);

// Run
app.listen(app.get('port'), ()=>{
    console.log('Server on Port 5000');
    console.log('NODE_ENV =', config.NODE_ENV);
    console.log('App listening on http://'+config.HOST+':'+config.PORT);
    console.log('ORACLE: - user: '+ config.USERDB ,'pass: '+config.PASS)
    // console.log(limit)

});
```

Para poder una insert, lo primero que se hace es un post, donde va permitir recibir datos que vienen desde el frontend al backend, por medio de una clase que se llama Routes.js en el fronted, creamos un fecth que nos permite enviar información al backend.

```
const BACKEND = "http://localhost:5000";
const url_addStudent = `${BACKEND}/registryStudent`;

export async function AddStudent(name, lastname, carnet, phone, direction, email, password){
    return fetch(url_addStudent, {
        method: "POST",
        headers: {
            Accept: "application/json",
            "Content-Type": "application/json",
        },
        body: JSON.stringify({
            nombre          : name,
            apellido        : lastname,
            carnet          : carnet,
            telefono        : phone,
            direccion       : direction,
            correo          : email,
            password        : password
        }),
    });
}
```

La función tiene como nombre AddStudent que es la encarga de agregar al estudiante, a la función se agrega un fecth, donde se envía como primer parametro la ruta del backend que en este caso viene siendo ‘regestryStudent’.

```
router.post('/registryStudent', async (req, res)=>{
```

Por medio del request, tiene como atributo el cuerpo que viene siendo todas las variables que trae desde el fronted, se inicializa una variable sql, que es la encargada de enviar a la función DB.Open, donde si es un insert, update, o algun dato que sea cambiado a la base de datos, de de ir true, mientras que si es una consulta debe de ir false.

```
router.post('/registryStudent', async (req, res)=>{

    var sql = `BEGIN insert_alumno ( '${req.body.nombre}','${req.body.a
    var result = await DB.Open(sql,[],true)

    res.status(200).send(JSON.stringify(result));
```

# BACKEND

## Getting Started with Create Node.js App

Crear proyecto de Node js

```
npm init -y
```

---

# DEPENDENCIAS

```
npm install express morgan oracledb cors
```

Por ultimo instalar la libería nodemon para que cada vez que se actualice un cambión lo vaya a realizar la appi.

```
npm install nodemon
```

# Execute
bash npm
start

## # CONFIGURACIONES

### JDK

Cambiar la versión de java

```
sudo update-alternatives --config java
```

Version de JDK 11 o 8

```
sudo gedit /etc/bash.bashrc
```

---

### Sqldeveloper

Configurar el SetJavaHome a la versión 11

# SetJavaHome /usr/lib/jvm/java-11-openjdk-amd64

```
sudo gedit ~/.sqldeveloper/21.2.1/product.conf
```

---

**Execute** Cargar imagen de Docker

```
sudo docker start oracle
sudo docker logs oracle
sudo docker ps
```

Abrir sqldeveloper

```
sh /opt/sqldeveloper/sqldeveloper.sh
```

---

### Datamodeler

Configurar el SetJavaHome a la versión 8

# SetJavaHome /usr/lib/jvm/java-8-openjdk-amd64

```
sudo gedit ~/.oraclesqldeveloperdatamodeler/21.4.1/product.conf
```

## Procedimiento Almacenado

```
-- Solicitamos todos los objetos que son procedimientos:
select *from user_objects where object_type='PROCEDURE';
-- Eliminar Procedure
DROP PROCEDURE name_procedure;
```

### Crear un Procedimiento Almacenado

```
CREATE OR REPLACE PROCEDURE nombre_procedimiento
(<param1> [IN|OUT|IN OUT] <type>,
 <param2> [IN|OUT|IN OUT] <type>, ...)

IS
-- Declaracion de variables locales
BEGIN
-- Sentencias
EXCEPTION
-- Sentencias control de excepcion
END nombre_procedimiento;
```

Cuando crea un procedimiento o una función, puede definir parámetros. Hay tres tipos de parámetros que se pueden declarar:

1) Parámetro de tipo IN: Estos tipos de parámetros se utilizan para enviar valores a procedimientos almacenados por lo tanto su valor de parámetro no puede ser reemplazado.

2) Parámetro de tipo OUT: Estos tipos de parámetros se utilizan para obtener valores de los procedimientos almacenados. Por consiguiente es similar a un tipo de retorno en funciones.

3) Parámetro IN OUT: Estos tipos de parámetros se utilizan para enviar valores aunque también para obtener valores en los procedimientos almacenados

### Llamada de un procedimiento almacenado

```
BEGIN
   insert_alumno (
    <param1>,
    <param2>, ...
    );
END;
```

### Crear una Funcion

```
CREATE [OR REPLACE] FUNCTION NOMBRE_FUNCION
[(
   param1 |IN|OUT|IN OUT| TIPO_DATO,
   param2 |IN|OUT|IN OUT| TIPO_DATO
), … ]
RETURN TIPO_DATO
IS | AS
--Declaración de variables locales
BEGIN
--Declaración de Sentencia
RETURN DATO;
   [EXCEPTION TIPO_EXCEPCION]
--Sentencias de control de excepción
END [NOMBRE_FUNCION];
```