PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- **Denominación del Programa de Formación:** TGO Análisis y Desarrollo de Sistemas de Información
- **Código del Programa de Formación:** 228106 V102
- **Nombre del Proyecto:** Construcción de un sistema de información que cumpla con los requerimientos del cliente en procesos que se lleven a cabo en el sector productivo del departamento de Caldas
- **Fase del Proyecto:** IMPLEMENTACIÓN
- **Actividad de Proyecto:** Seleccionar la alternativa de solución que cumpla con los requerimientos establecidos por el cliente
- **Competencia:** Construir el sistema que cumpla con los requisitos de la solución informática.
- **Resultados de Aprendizaje Alcanzar:** Realizar la codificación de los módulos del sistema y el programa principal, a partir de la utilización del lenguaje de programación seleccionado, de acuerdo con las especificaciones del diseño
- **Duración de la Guía:** 40 horas

2. PRESENTACIÓN

En esta guía los aprendices con el acompañamiento del instructor, desarrollarán ejemplos y proyectos utilizando herramientas tecnológicas de vanguardia con la arquitectura **API Rest**. Se desarrollará una aplicación que permita desarrollar habilidades específicas en el área Web integrando diversos componentes inicialmente desde el área del Backend.

Rest: Representational State Transfer
API: Application Programming Interface

.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- **Descripción de la(s) Actividad(es)**
  - **Actividades de aprendizaje**
    - Creación de Servicios para comunicación con el Backend
    - Utilización del ruteo (Routing) para movernos entre los diferentes componentes
    - Creación de Formularios
    - Creación de API Rest utilizando Express y Node
    - Utilización de base de datos para la persistencia de datos

- **Actividad de Reflexión inicial**

Actividad de reflexión inicial

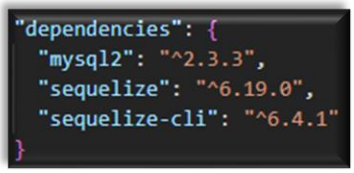
Antes de comenzar los diferentes proyectos, los aprendices deberán conocer la terminología básica de API Rest, complementando los conceptos fundamentales que se han visto en las guías pasadas. Para ello se deberán analizar los siguientes videos:

¿Que son las APIs y para qué sirven?
https://www.youtube.com/watch?v=u2Ms34GE14U

REST y RESTful APIs | Te lo explico en 5 minutos!
https://www.youtube.com/watch?v=JD6VNRdGl98

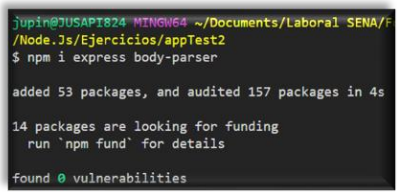
- **Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje**

En la guía anterior vimos configurar la parte inicial del proyecto incluyendo el ORM Sequelize para conectarnos con nuestra base de datos MySQL. Recordemos que las dependencias que se han instalado son las siguientes:

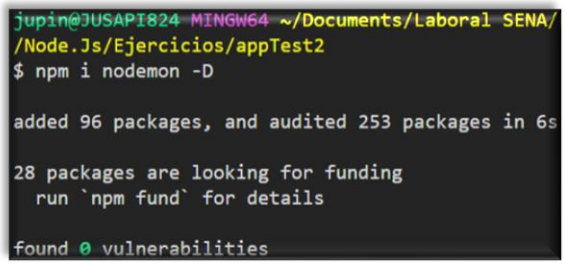


Continuando con la aplicación, seguiremos los siguientes pasos:

1. Instalar los paquetes: **body-parser** (para obtener los datos que se piden a través del body), **express** (que es el FrameWork de Node.js que usaremos) y **nodemon** (para que al momento de hacer un cambio en el código, se reinicie automáticamente el servidor)
   - `node i express body-parser`

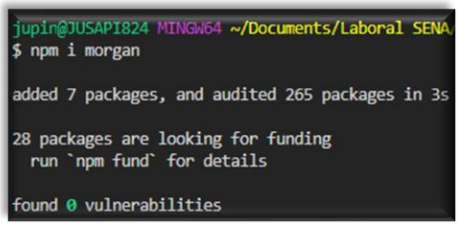


- `node i nodemon -D` (La opción -D es para instalar como una dependencia de desarrollo)

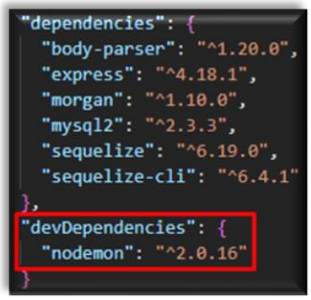


2. También instalaremos la dependencia **Morgan** que es un middleware para llevar un control de la información de las peticiones http que se realicen
    - Un **middleware** es una función que procesa datos antes de que el servidor los reciba

    `npm i morgan`



Después de Instalar estos paquetes, nuestras dependencias en el archivo package.json quedan de la siguiente manera (tener en cuenta que **nodemon** quedó en las dependencias de desarrollo):



3. Crear un archivo **server.js** con el siguiente código:

```
server.js > ...
// Including Dependencies

const express = require('express'); // Para requerir el frameWork Express de Node.js
const app = express(); // Instancia de Express
const bodyParser = require('body-parser'); // Permite leer el Cuerpo para analizarlo en un objeto Json
const morgan = require('morgan'); // Middleware que informa sobre las peticiones al servidor

// Settings
app.set('port', process.env.PORT || 4000); // Se setea el Puerto, toma 4000 si no está configurado en ENV

// Middlewares
app.use(bodyParser.urlencoded({extended:false})); // Para recibir datos desde un Formulario
app.use(bodyParser.json()); // Para que el servidor pueda recibir formato Json
app.use(morgan('dev')); // La opción dev da la información principal. Combined da más detalle

// Routes
// Se configura una ruta sencilla a través del método GET para probar
app.get('/', (req, res) => { // req=>request: llegan datos al server
    // res=>response: se envían los datos hacia el cliente
    //res.send('Hello ADSO !');// Se envía la respuesta en texto plano
    res.send({Title:'Hello ADSO !'}) // Envío en formato Json
});

// Starting de Server
app.listen(app.get('port'), () => { // Se inicia el servidor en el Puerto configurado
    console.log(`Server running on localhost:${app.get('port')}`);
});
```

4. Ejecutamos el comando: `nodemon server.js` y vemos como inicia nuestro servidor

```
jupin@JUSAPI824 MINGW64 ~/Documents/Laboral
$ nodemon server.js
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server running on localhost:4000
```

5. Otra opción para correr nuestro proyecto es editar el archivo **package.json** y en la sección de **scripts**, modificar el código como se muestra en la siguiente imagen:

```
package.json > ...
{
  "name": "apptest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "dev": "nodemon server.js"
  },
```

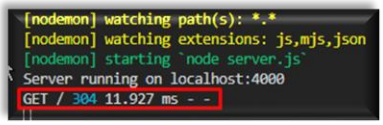
Para correr nuestro proyecto basta con ejecutar el siguiente código:

`npm run dev`

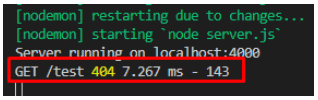
```
jupin@JUSAPI824 MINGW64 ~/Documents/Laboral
$ npm run dev

> apptest@1.0.0 dev
> nodemon server.js

[nodemon] 2.0.16
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server running on localhost:4000
```

6. Para testear nuestra ruta de prueba vamos al navegador y accedemos a la siguiente URL: http://localhost:4000/

Salida en el navegador



Información que brinda la dependencia **Morgan** (Tipo de petición, código de respuesta, tiempo de respuesta y tamaño)



http://localhost:4000/test

Si accedemos a esta url, nos arrojará un estado 404

7. Creamos una carpeta llamada: **api.** Luego dentro de la misma crearemos el archivo **users.js** con el siguiente código:

```
const db = require('../models');
const {Router} = require('express');
// Creamos el router para poder usar los verbos HTTP
const router = Router(); // Llamamos al método Router de Express

// req => request  => En request llegan los datos del body
// res => response => Se envían los datos hacia al cliente
router.get("/", (req, res) => {
    res.send({Title:'Hello ADSO!'});
});

module.exports = router;
```

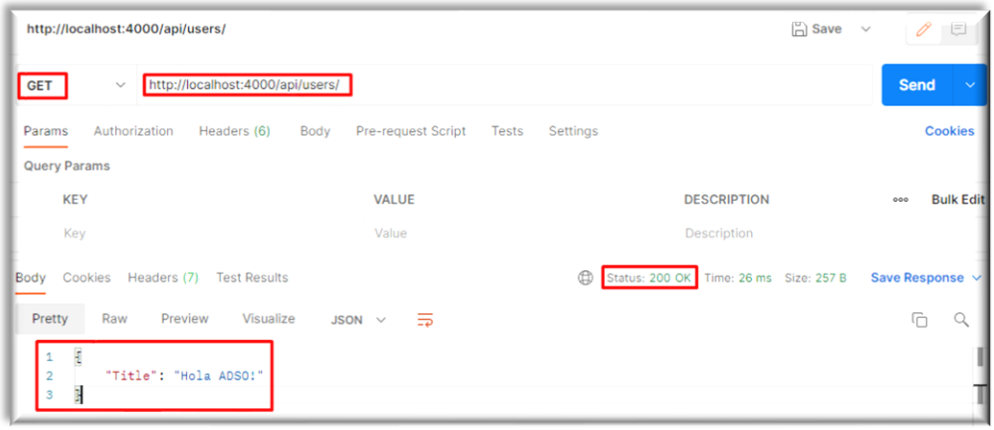
8. Editamos el archivo: **server.js** para mejorar el código en cuanto a la definición de la **rutas.** Para nuestro ejemplo, crearemos las rutas para todo lo que tiene que ver con usuarios:

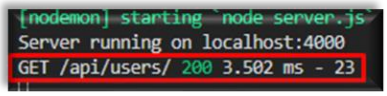
```
// Middlewares
app.use(bodyParser.urlencoded({extended:false})); // Para recibir datos desde un Formulario
app.use(bodyParser.json()); // Para que el servidor pueda recibir formato Json
app.use(morgan('dev')); // La opción dev da la información principal. Combined da más detalle

// Routes
app.use('/api/users', require('./api/users')); // Ruta para users
```

9. Editamos el archivo **users.js** para agregar la nueva ruta con la petición POST para poder guardar un usuario:

```
router.post('/new', async (req, res)=>{
    let name = req.body.name;
    let email = req.body.email;
    let password = req.body.password;
    try {
        await db.User.create({
            name,
            email,
            password,
        });
        res.status(200).send('Usuario creado');
    } catch (error) {
        res.status(400).send('Usuario no pudo ser creado');
    }
});
```

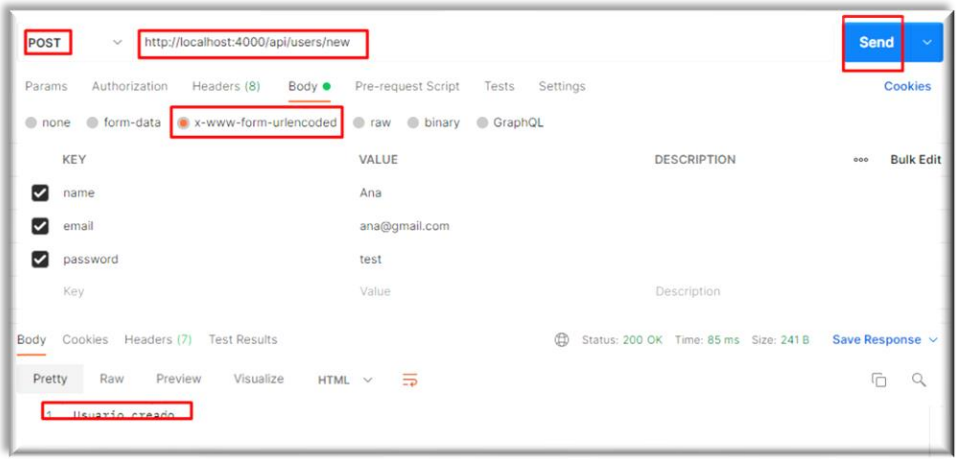
10. Abrimos **POSTMAN** y abrimos un nuevo request para probar el verbo GET y hacer el llamado correspondiente:





En la consola también podemos visualizar los detalles de la petición

11. Ahora vamos a probar el verbo POST y hacer el llamado correspondiente para poder ingresar un usuario



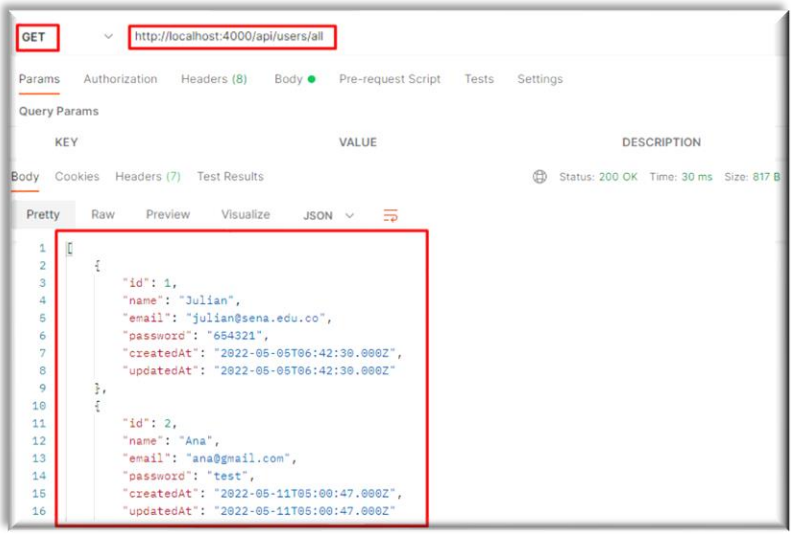
A continuación vemos en la consola, la sentencia SQL que se ejecuta en el servidor:

```
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Servidor corriendo en localhost:4000
Executing (default): INSERT INTO `Users` (`id`,`name`,`email`,`password`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?,?);
```

12. Agregamos el siguiente código a **users.js** con el fin de generar la ruta para traer **todos** los elementos:

```
router.get('/all', async (req,res) => {
    try {
        let users = await db.User.findAll();
        res.status(200).send(users);
    } catch (error) {
        res.status(400).send('No se pudieron obtener los usuarios')
    }
})
```

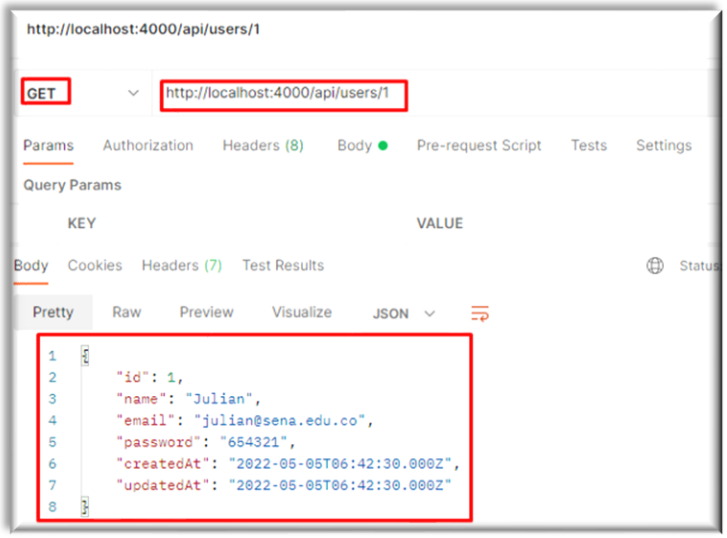
13. Probamos con POSTMAN la funcionalidad anterior:



14. Agregamos el siguiente código a **users.js** con el fin de generar la ruta para **traer un usuario** específico:

```
router.get('/:id', async (req,res) => {
    try {
        let id = req.params.id;
        let users = await db.User.findByPk(id);
        res.status(200).send(users);
    } catch (error) {
        res.status(400).send('No se pudo obtener el usuario')
    }
})
```

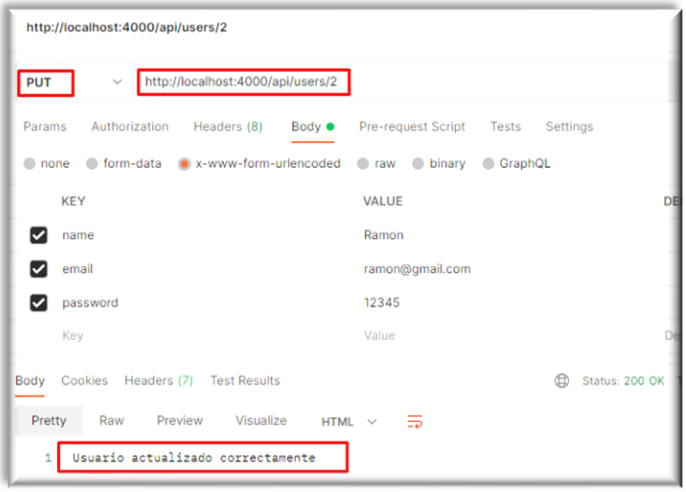
15. Probamos con POSTMAN la funcionalidad anterior:



16. Agregamos el siguiente código a **users.js** con el fin de generar la ruta para **actualizar un usuario** específico

```
router.put('/:id', async (req, res) =>{
    try {
        let id = req.params.id;
        let {name, email, password} = req.body;
        await db.User.update(
            {name, email, password},
            {
                where:{
                    id,
                }
            }
        );
        res.status(200).send("Usuario actualizado correctamente");
    } catch (error) {
        res.status(400).send("No se pudo actualizar el usuario");
    }
})
```

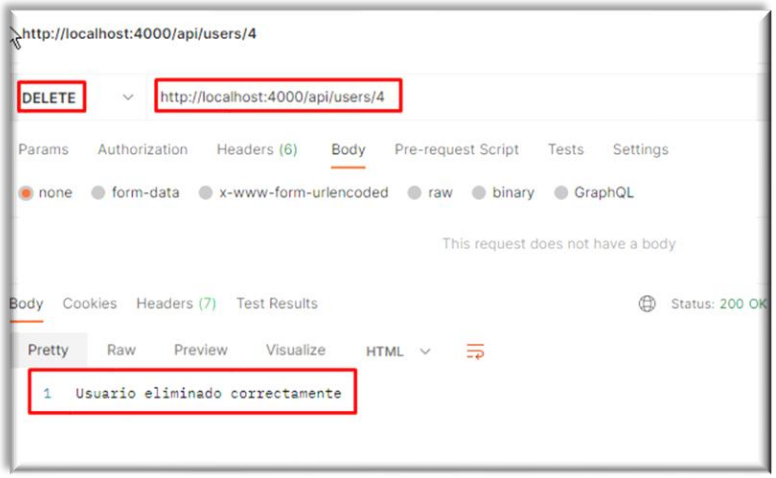
17. Probamos con POSTMAN la funcionalidad anterior:



18. Agregamos el siguiente código a **users.js** con el fin de generar la ruta para **eliminar un usuario** específico

```
router.delete('/:id', async (req, res) =>{
    try {
        let id = req.params.id;
        await db.User.destroy({
                where:{
                    id,
                }
            });
        res.status(200).send("Usuario eliminado correctamente");
    } catch (error) {
        res.status(400).send("No se pudo eliminar el usuario");
    }
});
```

19. Por último probamos con POSTMAN la funcionalidad anterior:



Podemos ver en la consola, las sentencias SQL que se ejecutan por Sequelize de lado del servidor:

```
[nodemon] starting `node server.js`
Servidor corriendo en localhost:4000
Executing (default): UPDATE `Users` SET `name`=?,`email`=?,`password`=?,`updatedAt`=? WHERE `id` = ?
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Servidor corriendo en localhost:4000
Executing (default): DELETE FROM `Users` WHERE `id` = '4'
```

- **Actividades de transferencia del conocimiento**
  Los aprendices deberán demostrar los conocimientos conceptuales y procedimentales que adquirieron a partir de las actividades de apropiación mediante el siguiente instrumento de evaluación:

Los aprendices deberán realizar cada una de las actividades propuestas para adquirir dominio en los conocimientos relacionados con los diferentes temas. Se evaluarán los conocimientos adquiridos mediante instrumentos de evaluación de Desempeño y Producto relacionados en la presente guía

- **Ambiente Requerido**
  Ambiente de SISTEMAS con conexión eléctrica e internet
- **Materiales**
  - Computadores (30)
  - Sillas (3)
  - Televisor (1)
  - Resma tamaño carta (1)
  - Marcadores (3)

- Lápiz (1)
- Lapicero (1)

4. ACTIVIDADES DE EVALUACIÓN

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento: **Evidencias de Desempeño:** Asistencia y participación activa en las diferentes actividades propuestas **Evidencias de Producto:** Respuestas y procedimiento de los talleres realizados	Crea la base de datos en el motor de base de datos seleccionado, siguiendo especificaciones técnicas del informe, según normas y protocolos de la empresa.	**Observación:** EXC_D_01 **Valoración del Producto:** EXC_P_01

5. GLOSARIO DE TÉRMINOS

Sistema de información: es un conjunto de elementos orientados al tratamiento y administración de datos e información, organizados y listos para su uso posterior, generados para cubrir una necesidad o un objetivo

Sistema operativo – Es un conjunto de programas que sirven para manejar un ordenador.

Software - El conjunto de programas, procedimientos y documentación asociado a un sistema informático.

Javascript: es un lenguaje de programación del lado del cliente que se utiliza con frecuencia en diseño WEB para generar efectos más complejos que no se puedan alcanzar usando HTML.

HTML: Siglas de las palabras inglesas: Hypertext Markup Language. Es decir, lenguaje de marcado de hipertexto. Lenguaje informático para crear páginas web. Conjunto de etiquetas o instrucciones que permiten estructurar el contenido de una web e incluir los hipervínculos o enlaces enlaces a otras páginas. Este lenguaje lo inventó en 1991 el Doctor Berners-Lee del CERN en Suiza.

HTTPS: Siglas de las palabras inglesas: HyperText Transfer Protocol Secure o versión segura del protocolo HTTP. Es el protocolo empleado para la transferencia de ficheros HTML cifrados que puedan contener información confidencial.

HTTP: siglas de las palabras inglesas: Hypertext Transfer Protocol. A saber en español: Protocolo de Transmisión de Hipertexto. Protocolo estándar de transferencia de hipertexto. Es decir: el protocolo de comunicaciones en el que está basado la Word Wide Web.

Script: es un archivo de órdenes o archivo de procesamiento por lotes. Es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.

MySQL: es un sistema de gestión de bases de datos de código abierto que, junto con PHP, permite darle a las páginas web cierto dinamismo, es decir, disponer de manera adecuada los datos solicitados por los navegadores. Es un sistema multiplataforma y su uso está tan extendido en las bases de datos que podría considerarse un estandar.

SEO (Search Engine Optimisation) Optimización en buscadores: técnica utilizada para asegurar que una página Web es compatible con los motores de búsqueda y así tener la posibilidad de aparecer en las posiciones más altas en los resultados de búsqueda.

Diseño web adaptable (responsive web design): se llama así al diseño web de aquellas páginas que se adaptan al tamaño de la pantalla o ventana en que se despliegan, por medio del uso de, idealmente, un solo documento HTML y un solo documento CSS. Esto permite hacer una sola página web para smartphones, phablets, tablets y PC.

Diagrama o Modelo Entidad Relación (DER): denominado por sus siglas en inglés, E-R "Entity relationship", o del español DER "Diagrama de Entidad Relación") es una herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información así como sus interrelaciones y propiedades

Bases de Datos (BD): es un banco de información que contienen datos relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o relación que busca ordenarlos y clasificarlos en conjunto.

6. REFERENTES BILBIOGRÁFICOS

- Documentos técnicos relacionados en la plataforma
- https://es.acervolima.com/cual-es-la-diferencia-entre-save-y-save-dev-en-node-js/
- https://www.desarrollo-web-br-bd.com/es/node.js/cual-es-la-diferencia-entre-save-y-save-dev/1046607843/
- https://code.tutsplus.com/es/tutorials/file-upload-with-multer-in-node--cms-32088
- https://enmilocalfunciona.io/entendiendo-cors-y-aplicando-soluciones/

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	Julian Salazar Pineda	Instructor	Centro de Procesos	14 Abril de 2022

			Industriales y Construcción	

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	**Cargo**	**Dependencia**	**Fecha**	**Razón del Cambio**
Autor (es)					