UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

Licenciatura en informática y tecnologías computacionales.

Centro de ciencias básicas.

Manual Técnico

César Eduardo López Baltazar

Jesús Aramis Acevedo Rivera

Alan Ellud Gonzales Fausto

Sixto Jesús Escobar Cervantes

Administracion y Diseño Web

Manual Técnico del API

Manual técnico del desarrollo de una Api para el control y manejo de una base de datos desarrollada con la tecnología MongoDB

Conexión de la base de datos

Se realizo una conexión básica a la base de datos mediante una URI encriptada en un archivo .env.

_Segmento de codigo **database.js**

```
const mongoose = require('mongoose');
require('dotenv').config(); // Esta línea carga las variables de entorno desde
el archivo .env

//Aqui se obtiene la URI del archivo .env
const URI = process.env.MONGO_URI;

//Se conecta a la base de datos
mongoose.connect(URI);

const connection = mongoose.connection;

//En caso de que la conexion sea correcta se indica con un mensaje
connection.once('open', () => {
    console.log('Base de datos conectada')
});
```

Enrutador

Se hiso uso de un archivo Router para poder administrar las rutas de la Api mar organizadamente. Esto nos permite organizar las rutas en dos tipos: las que reciben un código y las que no reciben un código. Al acceder a una ruta se ejecuta un método del archivo **producto.controller.js.**

_Segmento de código **producto.js**

```
//Rutas de producto producto.js
const { Router } = require('express');
const router = Router();

//Se obtienen los controlodores del archivo producto.controllers.js
const   {   getProductos,   getProducto,   createProducto,   updateProducto,
deleteProducto } = require('../controllers/producto.controller');

//Rutas para metodos que no requieren ID
router.route('/')
    .get(getProductos)      //Ruta para la consulta de todos los documentos
    .post(createProducto);  //Ruta para insertar un documento

//Rutas para metodos que requieren el codigo
router.route('/:codigo')
    .get(getProducto)       //Ruta para consultar un documento
    .put(updateProducto)    //Ruta para actualizar un documento
```

```
    .delete(deleteProducto);//Ruta para eliminar un documento
```

Controladores

Se desarrollaron 5 controladores para llevar a cabo los procesos básicos de Altas, Bajas Cambios y consultas a la base de datos.

Consulta General

El código para consultar todos los documentos simplemente hace la consulta a la base de datos y envía los resultados al cliente.

_Segmento de código **producto.controller.js** método **getProductos**

```
//Control para consultar todos los documentos
productoCtrl.getProductos = async (req, res) =>
{
    //Consulta los documentos
    const ListaProductos = await Producto.find();

    //Envia los documentos
    res.json(ListaProductos);
}
```

Alta

Para realizar una alta a la base de datos se recibe la información otorgada por el cliente para después realizar la alta a la base de datos.

_Segmento de código **producto.controller.js** método **createProducto**

```
//Control para crear un documento
productoCtrl.createProducto = async (req, res) =>
{
    try {
        //Se obtienen los datos por parte del cliente
       const { codigo, nombre, descripcion, precio, imgurl } = req.body;imgurl
        const ProductoNuevo = new Producto({
            Codigo: codigo,
            Nombre: nombre,
            Descripcion: descripcion,
            Precio: precio,
            ImgURL: imgurl
        });

        //Imprime los datos recibidos
        console.log(ProductoNuevo);

        //Inserta el documento
        await ProductoNuevo.save();

        //Responde al cliente con un mensaje
```

```
        res.json({ message: 'Producto Creado' });
    } catch (error) {
        console.error(error);
          res.status(500).json({  message:  'Error  del  servidor  al  crear  el
producto' });
    }
}
```

Consulta Particular

Para consultar un solo documento de la base de datos el servidor recibe le código del documento por parte del cliente, realizar la consulta y le devuelve al cliente la información obtenida.

_Segmento de código **producto.controller.js** método **getProducto**

```
productoCtrl.getProducto = async (req, res) =>
{
    try {
        //Se obtiene el codigo brindado por el cliente
        const codigo = req.params.codigo; // Asumiendo que el código está en
los parámetros de la solicitud
        console.log('El servidor recibio el codigo ', codigo);

        //Se realiza la consulta en la base de datos
        const producto = await Producto.findOne({ Codigo: codigo });

        //En caso de no existir manda mensaje de error
        if (!producto) {
             return res.status(404).json({ message: 'Producto no encontrado'
});
        }

        return res.json(producto);
    } catch (error) {
        console.error(error);
        return res.status(500).json({ message: 'Error del servidor' });
    }
}
```

Actualización

Para la actualización de un documento el servidor recibe el código del documento a modificar y la nueva información del documento posteriormente realiza la actualización del documento.

_Segmento de código **producto.controller.js** método **updateProducto**

```
//Control para actualizar un documento
productoCtrl.updateProducto = (req, res) =>
{
    //Obtiene los datos brindados por el cliente
    const { codigo, nombre, descripcion, precio, imgURL } = req.body;
```

```
    //Realiza la busqueda y actualiza el documento
    Producto.findOneAndUpdate(
        { Codigo: req.params.codigo }, // Buscar por el campo Codigo
        {
            Codigo: codigo,
            Nombre: nombre,
            Descripcion: descripcion,
            Precio: precio,
            ImgURL: imgURL
        },
        { new: true } // Para devolver el documento modificado en lugar del
original
    )
    .then(updatedProducto => {
        if (!updatedProducto) {
             return res.status(404).json({ message: 'Producto no encontrado'
});
        }
        res.json({ message: 'Producto Modificado', data: updatedProducto });
    })
    .catch(error => {
        console.error('Error al actualizar el producto:', error);
        res.status(500).json({ message: 'Error interno del servidor' });
    });
};
```

Eliminación

Para la eliminación simplemente se recibe el código del documento y posteriormente elimina el documento de la base de datos.

_Segmento de código **producto.controller.js** método **deleteProducto**

```
//Control para eliminar un documento
productoCtrl.deleteProducto = async (req, res) =>
{
    try {
        //Recibe el codigo por parte del cliente
        const codigo = req.params.codigo;

        // Buscar y eliminar el producto por código
        const resultado = await Producto.deleteOne({ Codigo: codigo });

        if (resultado.deletedCount === 0) {
             return res.status(404).json({ message: 'Producto no encontrado
para eliminar' });
        }

        res.json({ message: 'Producto Eliminado' });
    } catch (error) {
        console.error(error);
```

```
        res.status(500).json({ message: 'Error del servidor al eliminar el
producto' });
    }
}
```

Creación de la estructura de la Api

Los módulos con los que la Api funciona son Express, MongoDB y Nodemon para las pruebas en el desarrollo, esto haciendo uso del entorno de Node.js. Para la creación e instalación de esta estructura fue necesario realizar una serie de comandos en la consola del sistema operativo.

_Segmento de codigo **install.bash**

```
1. npm init
2. npm install express
3. npm install mongodb
4. npm install -g nodemon
```

Al crear instalar los módulos necesarios se nos creó el archivo Json con las propiedades del Api

_Segmento de código **pakage.json**

```
{
  "name": "2_backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "npx nodemon src/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "16": "^0.0.2",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "mongoose": "^8.0.2"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

Siguiendo estos pasos se logro crear una Api funcional que permite interactuar con la base de datos satisfactoriamente. Después el Api fue incorporado al FrontEnd del sistema logrando así un sistema completo y funcional.