**Categorías y productos**

Aquí cubriremos varios temas como:

1. Tareas
2. CRUD de categorías y productos
3. Relaciones
4. Populate
5. Búsquedas
6. Despliegues a producción

Creación de servicios para categorías

Basado en el código para el usuario creamos los servicios /categoría para get y post y el servicio /categoría para put, get y delete.

**Integrando contenido de otras tablas – populate**

En categoría tenemos el id del usuario que crea o modifica una categoría, pero ahora queremos su nombre o datos, para esto usaremos **populate. El populate expande los valores viniculados al \_id usando el esquema, para esto se debió insertar el id de referencia en el esquema que etamos trabajando. Queda así:**

**Para la petición**

app.get('/categoria', verificaToken, (req,res) => {

    Categoria.find({}) //puedes ser .find({}, (err,categoriasDB)) pero no se podrian agregar otros funciones estras

    .populate('usuario', 'nombre email')  //Si hay otros id de otras tablas, solo se pone de nuevo el populate

    .sort('descripcion'),       //Ordena las categorias alfabeticamente por descripcion

    .exec((err, categoriaDB) =>

    {

        if(err)

Desde la definición de esquema:

const mongoose = require('mongoose');

//let usuario = require('./usuario');

let Schema = mongoose.Schema;

let categoriaSquema = new Schema(

    {

        descripcion:{

            type: String,

            unique: [true,'Solo puede haber un producto con esa descripción'],

            required: [true, 'Se requiere una descripción']

        },

        usuario: {

            type: Schema.Types.ObjectId, ref: 'usuario'

        }

    });

    module.exports = mongoose.model('categoria',categoriaSquema);

Es importante que en el populate(‘esquema’, ‘param1 param2’) se utilice el nombre tal cual se exportó el esquema, en el caso de arriba sería populate(‘categoria’, ‘descripcion’) NO populate(‘Categoria’, ‘descripcion’) u alguna otra variación, el segundo parámetro es para que devuelva solo los valores deseados.

Cuando necesitamos mandara parámetros a la función y queremos usar populate podemos hacer:

app.get('/producto/:id', (req,res) =>{

    let id = req.params.id;

    Producto.findById(id)  //Si necesito mandar parametro, lo puede hacer al definir la función y se recibe los resultados y se implementa en el exec

    .populate('usuario', 'nombre email')

    .populate('categoria', 'descripcion')

    //.sort('nombre')

    .exec((err, productoDB)=>{

        if(err)

        {

            return res.status(500).json({

pasando los parámetros y ejecutando posteriormente.

**BÚSQUEDA**

Para esto utilizaremos la función ObjetoModelo.find({parametro: valor}) en un servicio app.get(‘/producto/buscar/:termino’,(etc)=>{}) donde termino lo obtenemos de let temino= req.params.termino

Esto realiza una búsqueda como el nombre tal cuál esta registrado, para una mejor búsqueda utilizaremos una expresión regular de esta manera:

app.get ('/producto/buscar/:termino',verificaToken, (req, res)=>{

    let termino = req.params.termino;

    let regexp = new RegExp(termino, 'i'); // Crea una expresion regular, usando el termio, la i es para que no tome en curnta las mayusculas y minusculas

     Producto.find({nombre: regexp})

    .populate('categoria', 'descripcion')

Y continuamos con el código de forma normal.

**Despliegue a heroku y respaldo en git Hub.**

Para poder ver los cambio en producción, debemos subir nuestro proyecto , en heroku que es el servidor que estamos utilizando y también en git hub como respaldo.

Git status comprobamos los cambios

Git add . //Guardamos los cambios en el stage

Git commit -m “servicios rest de catalago y productos con búsqueda incluida”

Git push heroku master // Subimos y desplegamos a heroku

Git push master //Subimos a git hub para respladar

Git tag -a v1.0.2 -m “servicios rest de catalago y productos con búsqueda incluida” // Creamos el tag

Git push –tags //Subimos el tag a git hub