Trabajo Teórico / Práctico de Introducción a Express.js

1)

En el archivo package.json figuran 5 dependencias de desarrollo y 8 dependencias de aplicación:

Dependencias de Desarrollo:

"eslint": "^8.8.0", //herramienta de detección de errores.

"eslint-config-prettier": "^8.3.0", // herramienta de detección de errores.

"eslint-plugin-prettier": "^4.0.0", // herramientas de detección de errores.

"nodemon": "^2.0.15", // Recarga el servidor cuando se graba algun cambio en el codigo.

"prettier": "^2.5.1" // da formato homogeneo al código.

Dependencias de aplicación:

"@hapi/boom": "^9.1.4", // ayuda a capturar errores y mostrarlos de forma mas amigable.

"cors": "^2.8.5", // permite ejecucuión en entornos no monolíticos, es decir, permite que la aplicación pueda ser utilizada por plataformas externas al propio desarrollo.

"crypto": "^1.0.1", // se utiliza para encriptar información.

"dotenv": "^16.0.0", // permite configurar nuestro entorno de desarrollo.

"express": "^4.17.2", // carga el framework de la aplicación.

"joi": "^17.6.0", // valida que los datos ingresados tengan el contenido adecuado.

"jsonwebtoken": "^8.5.1", // maneja los tokens para transmitir peticiones seguras, se usa para poder mantener una sesion iniciada, ya que las api no manejan el concepto de cesion.

"mssql": "^8.0.1" // orm para manejar la base de datos.

2)

Posee 3 scripts de inicio definidos en package.json:

"scripts": {

"dev": "nodemon index.js", // se ejecuta durante el desarrollo, para refrescar el servidor ante cada cambio realizado y guardado en el código.

"start": "node index.js", // se ejecuta para probar o poner en producción el proyecto.

"lint": "eslint" // se lanza para analizar el código y detectar errores.

},

3)

Carga las variables necesarias para conectar a la base de datos.

4)

Routes: Definición de los endpoints.

Controllers: Lógica de la aplicación.

Services: Se encuentran los servicios y recursos de la aplicación.

Middlewares: Contiene las politicas relacionadas con las validaciones.

Schemas: Tiene información sobre la estructura de la base de datos.

5)

```
// toma id desde el endpoint que es optativo
static async get(id) {
// PRUEBA.
  try {
// consulta a la db si el id es null trae todos los registros, de lo contrario solo el del id.
    let qry = "SELECT * FROM vwGetProducts WHERE idproduct = CASE WHEN @id IS NULL THEN idproduct ELSE @id END";
// espera la conexión a la base de datos para poder hacer la consulta.
    let pool = await DB.connect();
// espera la respuesta el request de la conexión a la base de datos con los datos suministrados.
    let products = await pool.request().input('id', sql.Int, id).query(qry);
// si los resultados de la consulta son mayores a 0, entonces los muestra.
    if(products.rowsAffected > 0) return products.recordset;
// si no hay registros retorna null "[]".
    else return [];
  }
// SI FALLA
  catch (err) {
// envía error.
```

```
    throw 'Error inesperado'
 }
```

6)

// Toma el id del moviemiento que se envía.

```
delete: async (id) => {
```

// conecta a la base de datos.

```
    return DB.connect().then((pool) => {
```

// retorna el id del producto y la cantidad que movió el movimiento solicitado.

```
      return pool.request().input('id', sql.Int, id).query('SELECT idproduct, quantity FROM movements WHERE idmovement = @id')
```

// luego resta la cantidad que figura en el movimiento seleccionado a la cantidad que figura en stock del producto afectado.

```
      .then((data) => {
        pool.request().input('idproduct', sql.Int, data.recordset[0].idproduct).input('stock', sql.Int, data.recordset[0].quantity).query('UPDATE products SET stock = stock - @stock WHERE idproduct = @idproduct')})
```

// luego elimina el movimiento.

```
        .then(() => {
          return pool.request().input('id', sql.Int, id).query('DELETE FROM movements WHERE idmovement = @id') })
```

// retorna el resultado de la eliminación del movimiento.

```
          .then((result) => {return result})
            .catch(function(err) { throw err.message });
    })
```

((Creo según lo interpretado: la acumulación de movimientos se podría usar para tener un seguimiento o un LOG de lo que sucede con los productos, al eliminar un movimiento (en el caso de un movimiento de una venta) el producto debería sumarse la cantidad en stock, ya que se "deshace" una salida. Este código podría formar parte del manejo de stock del lado de los proveedores, es decir, al "deshacer" una compra a un proveedor, debería restarse el stock del producto en cuestión)).

7)

Boom permite enviar errores con información contextualizada y mas completa.

En **throw boom.badRequest(err)** captura el error de ejecución de **function(ret){}**, y responde con un error de ejecución.

En el caso de **throw boom.unauthorized("invalid access")** captura el error del resultado de la función, si bién la funcion se ejecutó correctamente, el resultado no es válido, es decir el resultado es null, es válida la ejecución pero la respuesta en nula y por lo tanto responde una desautorización.

El beneficio de utilizar boom es poder lograr un mayor grado de detalle en el error, envía un marco contextual donde se encuentra el error, mas allá del error mismo. Con esa información en forma de json y cada item puede ser capturado y tratado independientemente.

8)

El token es utilizado para validar la autenticidad de la persona que esta intentando interactuar con los servicios.

La generación de dicho token depende del archivo config/token.js que contiene la clave privada o secreta para generar el las claves publicas que seran las que viajen cifradas en las cabeceras de cada solicitud.

Se genera en controllers/auth.controller.js.

A travez del middlewares/validator.handler.js se controla en cada request.

9)

//idcategory debe ser un entero positivo menor a 255, no debe ser nulo.

idcategory : Joi.number().integer().min(0).max(255).required(),

//denomination debe ser un texto de mas de 10 caracteres y menos de 255, no debe ser nulo

denomination: Joi.string().min(10).max(255).required(),

//additional_info es un campo opcional de texto con maximo de 100 caracteres

additional_info: Joi.string().min(0).max(100).optional(),

//price es un campo numérico con 2 decimales

price: Joi.number().precision(2),

//stock debe ser un entero positivo menor a 100.

stock : Joi.number().integer().min(0).max(100)

10)

next() indica que una vez ejecutada una funcion middleware, prosiga con la ejecución del resto del código. Es lo esencial del concepto de middleware, es una ejecución intermedia sin capturar el flujo de ejecución, permitiendo continuar con las demás tareas pendientes de proceso.