Manejo de errores con Boom

De acuerdo a lo que hemos visto en el curso, hay que tener en cuenta que **TODO** puede fallar, es la razón por la cual se trabaja tanto el tema de aprender a manejar y capturar errores, dado que por cómo funciona Node si un error no es capturado detendrá toda la aplicación.

Así podemos implementar el error en le middleware:

```
/* eslint-disable no-console */
const logErrors = (err, req, res, next) => {
  console.info('1. logErrors');
  console.error(err);
  next(err);
}

const errorHandler = (err, req, res, next) => {
  console.log('2. errorHandler');
  res.status(500)
  .json({
    message: err.message,
    stack: err.stack
  });
  next(err);
}

const boomErrorHandler = (err, req, res, next) => {
  console.log('3. boomErrorHandler');
  if(err.isBoom){
    const { output } = err;
    res.status(output.statusCode)
    .json(output.payload);
  }
  next(err);
}

module.exports = {
  logErrors,
  errorHandler,
  boomErrorHandler
};

```

Realizar la importación en el index:

```
const express = require('express');
const { boomErrorHandler, errorHandler, logErrors } = require(
'./middleWares/error.handler');
const routersApi = require('./routes')
const app = express();
const port = 3021;

app.use(express.json())

app.get('/',(req, res)=>{
  res.writeHead(200,{'Content-Type':'html'})
  res.write('<div>Hello World</div>')
  res.end()
})

routersApi(app)

app.use(logErrors);
app.use(boomErrorHandler);
app.use(errorHandler);

app.listen(port, ()=>{
  // eslint-disable-next-line no-console
  console.log(`Server up and listening on port ${port}`)
})

```

Para luego implementarlos en services:

```
const fakerJs = require('@faker-js/faker');
const boom = require('@hapi/boom');
const { v4 } = require('uuid');
const uuidv4 = v4;

class ProductServices {

  constructor() {
    this.products = this.generate(100)
  }

  generate (size) {
    const limit = size ? size : 100
    const fakedProducts = [];
    for(let i = 0; i < parseInt(limit); i++){
      fakedProducts.push({
        id: uuidv4(),
        name: fakerJs.faker.commerce.product(),
        price: parseInt(fakerJs.faker.commerce.price(), 10),
        image: fakerJs.faker.image.imageUrl(),
        isBlocked: fakerJs.faker.datatype.boolean(),
      });
    }
    return fakedProducts;
  }

  async create (product) {
    return new Promise((resolve, reject) => {
      try{
        setTimeout(()=>{
          this.products.push(product);
          resolve(this.findOne(product.id));
        },1000);
      }catch(e){
        reject(e);
      }
    })
  }

  async find (size) {
    return new Promise((resolve, reject)=>{
      try{
        setTimeout(()=>{
          let counter = 0
          const limit = size ? parseInt(size) : 100
          const results = this.products.filter(p=>{
            if(counter < limit) {
              counter++;
              return p;
            }
          });
          resolve(results);
        },2000);
      }catch(e){
        reject(e)
      }
    });
  }

  async findOne (id) {
    return new Promise((resolve, reject)=>{
      try{
        setTimeout(()=>{
          const result = this.products.filter(p => {
            if(id === p.id) return p;
          })
          if(result.length===0)
            reject(boom.notFound('Product not found'));
          else if(result[0].isBlocked)
            reject(boom.conflict("The product is blocked"));
          else resolve(result[0]);
        },1000);
      }catch(e){
        reject(e);
      }
    })
  }

  async update (product) {
    return new Promise((resolve, reject)=>{
      try{
        setTimeout(()=>{
          const index = this.products.findIndex(p=>p.id===product.id);
          if(index < 0) reject(boom.notFound('Product not found'));
          this.products.map(p=>{
            if(p.id === product.id){
              p.name = product.name ? product.name : p.name;
              p.price = product.price ? product.price : p.price;
              p.image = product.image ? product.image : p.image;
            }
          })
          resolve(this.products.filter(p=>{
            if(p.id === product.id) return p;
          }));
        },1000);
      }catch(e){
        reject(e)
      }
    })
  }

  async delete (id) {
    return new Promise((resolve, reject)=>{
      try{
        setTimeout(()=>{
          const index = this.products.findIndex(p=>p.id===id);
          if(index < 0) reject(boom.notFound('Product not found'));
          this.products = this.products.splice(index, 1);
          resolve('Product deleted successfully');
        },1000);
      }catch(e){
        reject(e)
      }
    })
  }

}

module.exports = ProductServices;

```