Estando en la ubicaion del proyecto:

npm init -y

Con lo cual se crea el archivo package.jon

Además se crean los siguientes archivos:

A screenshot of a computer

Description automatically generated

Para configurar el gitignore ir a gitignore.io y escoger el lenguaje y el sistema operativo.

.editorconfig

*# top-most EditorConfig file*

root = true

*# Unix-style newlines with a newline ending every file*

[\*]

charset = utf-8

indent\_style = space

indent\_size = 2

end\_of\_line = lf

insert\_final\_newline = true

trim\_trailing\_whitespace = true

[\*.md]

max\_line\_length =off

trim\_trailing\_whitespace = false

*# Indentation override for all JS under lib directory*

[\*.js]

quote\_style =single

*# Matches the exact files either package.json or .travis.yml*

[{package.json,.travis.yml}]

indent\_style = space

indent\_size = 2

.eslintrc.json

{

  "parserOptions": {

    "ecmaVersion": 2018

  },

  "extends": ["eslint:recommended", "prettier"],

  "env": {

    "jest": true,

    "es6": true,

    "node": true

  },

  "rules": {

    "no-console": "warn"

  }

}

Package.json

{

  "name": "my-store",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "dev": "nodemon index.js",

    "start": "nodemon index.js",

    "lint": "eslint"

  },

  "keywords": [],

  "author": "",

  "license": "ISC"

}

Descargar librerias :

npm i nodemon eslint eslint-config-prettier eslint-plugin-prettier prettier -D

**npm run dev To run server**

REST: Representational state transfer

npm i @faker-js/faker

app.get("/products", (*req*, *res*) => {

*const* products = [];

*const* { size } = req.query;

*const* limit = parseInt(size, 10) || 100;

  for (*let* i = 0; i < limit; i++) {

    products.push({

      name: faker.commerce.productName(),

      price: parseInt(faker.number.int(), 10),

      Image: faker.image.url(),

    });

  }

  res.json(products);

});

<http://localhost:3000/products?size=2>

SRP: Single Resposabilties Principles

In the main directory we create a file named index.js which contains the server:

*const* express = require("express");

*const* routerApi = require("./routes");

*const* app = express();

*const* port = 3000;

app.use(express.json());

app.get("/", (*req*, *res*) => {

  res.send("Hello my server is running");

});

app.get("/new-route", (*req*, *res*) => {

  res.send("Hi, I am a new route!");

});

app.get("/categories/:categoryId/products/:productId", (*req*, *res*) => {

*const* { categoryId, productId } = req.params;

  res.json({

    categoryId,

    productId,

  });

});

routerApi(app);

app.listen(port, () => {

  console.log(`Server is running on port ${port}`);

});

This line:

app.use(express.json());

is a middleware which allows to display server answer (res.json) after doing any request

The server is listening an app which has the root: “/api/v1” and the endpoints: products, users and categories ( all this is defined in the index file inside the router folder(

This code:

app.get("/categories/:categoryId/products/:productId", (*req*, *res*) => {

*const* { categoryId, productId } = req.params;

  res.json({

    categoryId,

    productId,

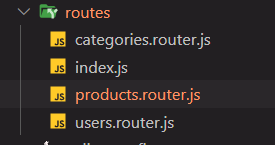
  });

});

Allow this if we write this in the browser :



In the root we create a folder named routes



Categories.router, products.router and user.router contains the controller with the verbs, for example this could the code in the file products.router:

*const* express = require("express");

*const* { faker } = require("@faker-js/faker");

*const* router = express.Router();

router.get("/", (*req*, *res*) => {

*const* products = [];

*const* { size } = req.query;

*const* limit = parseInt(size, 10) || 100;

  for (*let* i = 0; i < limit; i++) {

    products.push({

      name: faker.commerce.productName(),

      price: parseInt(faker.number.int(), 10),

      Image: faker.image.url(),

    });

  }

  res.json(products);

});

router.post("/", (*req*, *res*) => {

*const* body = req.body;

  res.json({

    message: "created successfully",

    data: body,

  });

});

router.get("/:id", (*req*, *res*) => {

*const* { id } = req.params;

  res.json({

    id,

    name: `Product ${id}`,

    price: 1000,

  });

});

router.patch("/:id", (*req*, *res*) => {

*const* { id } = req.params;

*const* body = req.body;

  res.json({

    message: "updated successfully",

    data: body,

    id,

  });

});

router.get("/filter", (*req*, *res*) => {

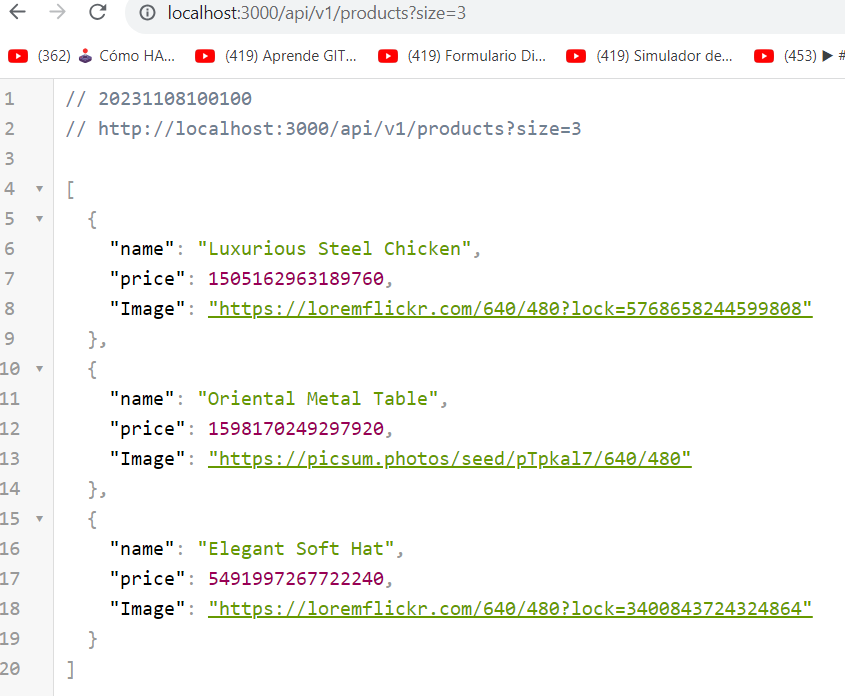
  res.send("<h1>Soy un Filter</h1>");

});

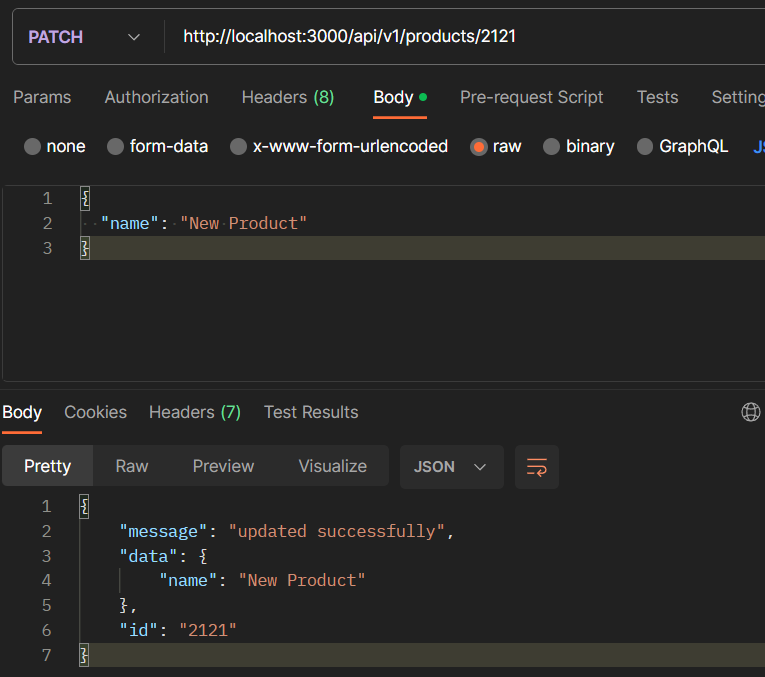
*module*.*exports* = router;

This API still does not have an DB so if wan to simulate read or get the data we use the get verb:

By default it generates 100 productos but we can specify how many with a query:



Example using the patch:



All this controllers are used in another index.js file which is in this same folder:

*const* express = require("express");

*const* productsRouter = require("./products.router");

*const* usersRouter = require("./users.router");

*const* categoriesRouter = require("./categories.router");

*const* routerApi = (*app*) => {

*const* router = express.Router();

  app.use("/api/v1", router);

  router.use("/products", productsRouter);

  router.use("/users", usersRouter);

  router.use("/categories", categoriesRouter);

};

*module*.*exports* = routerApi;

We use “api/v1” for further changes.

For example the line of code:

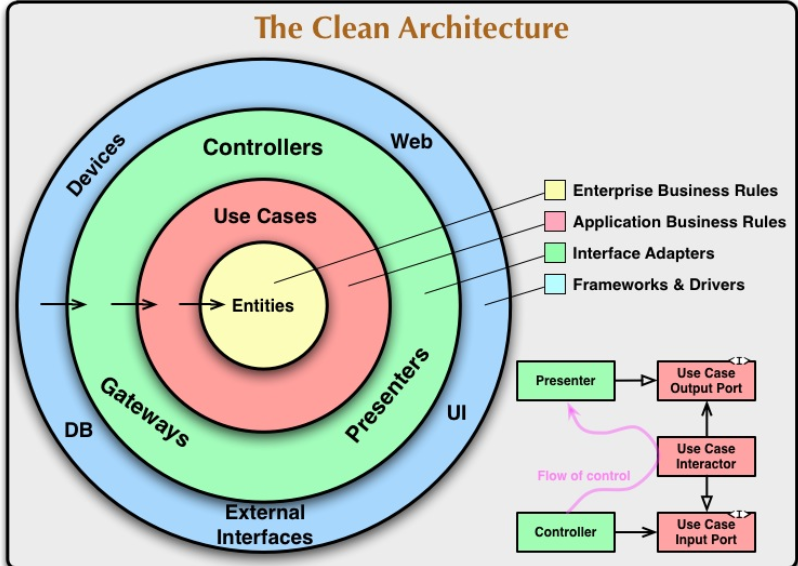
router.use("/products", productsRouter);

I think the logical order is:

Create a router folder with index.js and the controllers: products.router.js users.router.js and categories.router.js

After create the index.js in the main root and there coded the server.

Now in order to follow the clean arquitecture technique:



Let’s refactorize the code:

Create a new folder: 

Inside product.service.js write next code:

*const* { faker } = require("@faker-js/faker");

*class* ProductServices {

*constructor*() {

*this*.products = [];

*this*.generate();

  }

  generate() {

*const* limit = 5;

    for (*let* i = 0; i < limit; i++) {

*this*.products.push({

        id: faker.string.uuid(),

        name: faker.commerce.productName(),

        price: parseInt(faker.number.int(), 10),

        Image: faker.image.url(),

      });

    }

  }

*async* create(*data*) {

*const* newProduct = {

      id: faker.string.uuid(),

      ...data,

    };

*this*.products.push(newProduct);

    return newProduct;

  }

*async* find() {

    return *this*.products;

  }

*async* findOne(*id*) {

    return *this*.products.find((*item*) => item.id === id);

  }

*async* update(*id*, *changes*) {

*const* index = *this*.products.findIndex((*item*) => item.id === id);

    if (index === -1) {

      throw new Error("Product not found");

    }

*const* product = *this*.products[index];

*this*.products[index] = { ...product, ...changes };

    return *this*.products[index];

  }

*async* delete(*id*) {

*const* index = *this*.products.findIndex((*item*) => item.id === id);

    if (index === -1) {

      throw new Error("Product not found");

    }

*this*.products.splice(index, 1);

    return { id };

  }

}

*module*.*exports* = ProductServices;

And refactorize product.router.js:

*const* express = require("express");

*const* ProductServices = require("./../services/product.service");

*const* router = express.Router();

*const* service = new ProductServices();

router.get("/", *async*(*req*, *res*) => {

*const* products = await service.find();

  res.json(products);

});

router.post("/", *async*(*req*, *res*) => {

*const* body = req.body;

*const* newProduct = await service.create(body);

  res.status(201).json(newProduct);

});

router.patch("/:id", *async*(*req*, *res*) => {

*const* { id } = req.params;

*const* body = req.body;

*const* product = await service.update(id, body);

  res.json(product);

});

router.delete("/:id", *async* (*req*, *res*) => {

*const* { id } = req.params;

*const* rst = await service.delete(id);

  res.json(rst);

});

router.get("/:id", *async* (*req*, *res*) => {

*const* { id } = req.params;

*const* product = await service.findOne(id);

  res.json(product);

});

router.get("/filter", (*req*, *res*) => {

  res.send("<h1>Soy un Filter</h1>");

});

*module*.*exports* = router;

To simulate a delay in a process lest change the find method:

*async* find() {

    return new Promise((*resolve*, *reject*) => {

      setTimeout(() => {

        resolve(*this*.products);

      }, 5000);

    });

  }

Besides we can improve error handling using try-catch, for example:

router.patch("/:id", *async* (*req*, *res*) => {

  try {

*const* { id } = req.params;

*const* body = req.body;

*const* product = await service.update(id, body);

    res.json(product);

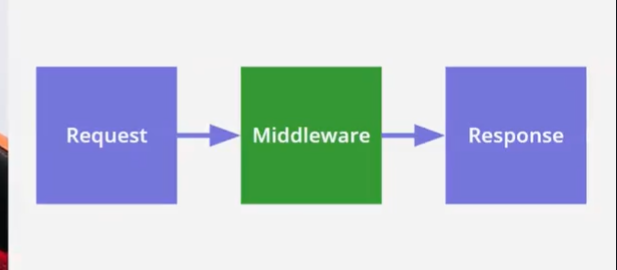
  } catch (error) {

    res.status(404).json({ error: error.message });

  }

});

Middleware:

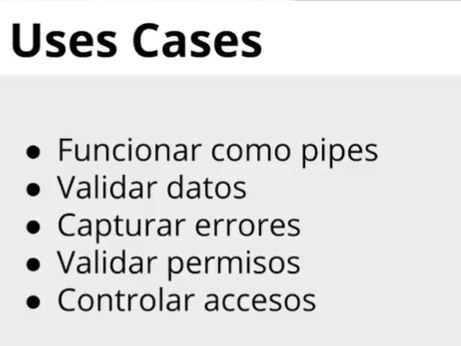




A computer code on a black background

Description automatically generated

Pipes: when we chain several middleware



Let’s create a folder for middleware: A black background with red and blue text

Description automatically generated

And code:

*const* logErrors = (*err*, *req*, *res*, *next*) => {

  console.error(err.stack);

  next(err);

};

*const* errorHandler = (*err*, *req*, *res*, *next*) => {

  res.status(err.status || 500).json({

    message: err.message,

  });

};

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

Middleare are called after the router

Index.js

*const* express = require("express");

*const* routerApi = require("./routes");

*const* { logErrors, errorHandler } = require("./middlewares/error.handler");

*const* app = express();

*const* port = 3000;

app.use(express.json());

app.get("/", (*req*, *res*) => {

  res.send("Hello my server is running");

});

routerApi(app);

app.use(logErrors);

app.use(errorHandler);

app.listen(port, () => {

  console.log(`Server is running on port ${port}`);

});

And now we can refactorize the get by id:

router.get("/:id", *async* (*req*, *res*, *next*) => {



  try {

*const* { id } = req.params;

*const* product = await service.findOne(id);

    res.json(product);



  } catch (error) {

    next(error);

  }

});

In this way when an error occurs using this endpoint the try.catch executes de error middleware

Handle errors with Boom

npm i @hapi/boom

in product.service.js :

*const* boom = require("@hapi/boom");

and we can use it like this:

*async* findOne(*id*) {

*const* product = *this*.products.find((*item*) => item.id === id);

    if (!product) {

      throw boom.notFound("Product not found");

    }

    return product;

  }

But besides we need to create another middleware in errorHandler.js:

*const* boomErrorHandler = (*err*, *req*, *res*, *next*) => {

  if (err.isBoom) {

*const* { output } = err;

    res.status(output.statusCode).json(output.payload);

  }else{

    next(err);

  }

};

In product.router change the endpoints codes with try-catch blocks:

router.patch("/:id", *async* (*req*, *res*, *next*) => {

  try {

*const* { id } = req.params;

*const* body = req.body;

*const* product = await service.update(id, body);

    res.json(product);

  } catch (error) {

    next(error);

  }

});

router.delete("/:id", *async* (*req*, *res*,*next*) => {

  try {

*const* { id } = req.params;

*const* rst = await service.delete(id);

    res.json(rst);

  } catch (error) {

    next(error);

  }

});

router.get("/:id", *async* (*req*, *res*, *next*) => {

  try {

*const* { id } = req.params;

*const* product = await service.findOne(id);

    res.json(product);

  } catch (error) {

    next(error);

  }

});

And the main index.js

*const* express = require("express");

*const* routerApi = require("./routes");

*const* {

  logErrors,

  errorHandler,

  boomErrorHandler,

} = require("./middlewares/error.handler");

*const* app = express();

*const* port = 3000;

app.use(express.json());

app.get("/", (*req*, *res*) => {

  res.send("Hello my server is running");

});

routerApi(app);

app.use(logErrors);

app.use(boomErrorHandler);

app.use(errorHandler);

app.listen(port, () => {

  console.log(`Server is running on port ${port}`);

});

The product.service.js file finally looks like this:

*const* { faker } = require("@faker-js/faker");

*const* boom = require("@hapi/boom");

*class* ProductServices {

*constructor*() {

*this*.products = [];

*this*.generate();

  }

  generate() {

*const* limit = 5;

    for (*let* i = 0; i < limit; i++) {

*this*.products.push({

        id: faker.string.uuid(),

        name: faker.commerce.productName(),

        price: parseInt(faker.number.int(), 10),

        Image: faker.image.url(),

        isBlock: faker.datatype.boolean(),

      });

    }

  }

*async* create(*data*) {

*const* newProduct = {

      id: faker.string.uuid(),

      ...data,

    };

*this*.products.push(newProduct);

    return newProduct;

  }

*async* find() {

    return new Promise((*resolve*, *reject*) => {

      setTimeout(() => {

        resolve(*this*.products);

      }, 5000);

    });

  }

*async* findOne(*id*) {

*const* product = *this*.products.find((*item*) => item.id === id);

    if (!product) {

      throw boom.notFound("Product not found");

    }

    if (product.isBlock) {

      throw boom.conflict("Product is blocked");

    }

    return product;

  }

*async* update(*id*, *changes*) {

*const* index = *this*.products.findIndex((*item*) => item.id === id);

    if (index === -1) {

      throw boom.notFound("Product not found");

    }

*const* product = *this*.products[index];

*this*.products[index] = { ...product, ...changes };

    return *this*.products[index];

  }

*async* delete(*id*) {

*const* index = *this*.products.findIndex((*item*) => item.id === id);

    if (index === -1) {

      throw boom.notFound("Product not found");

    }

*this*.products.splice(index, 1);

    return { id };

  }

}

*module*.*exports* = ProductServices;

Data validation with joi

Npm I joi

First that all create a new folder and a new file: 

Product.schema.js code:

*const* Joi = require("joi");

*const* id = Joi.string().regex(

  /^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}$/,

);

*const* name = Joi.string().min(3).max(15);

*const* price = Joi.number().integer().min(10);

*const* image = Joi.string().uri();

*const* createproductSchema = Joi.object({

  name: name.required(),

  price: price.required(),

  image: image.required(),

});

*const* updateproductSchema = Joi.object({

  name: name,

  price: price,

  image: image,

});

*const* getproductSchema = Joi.object({

  id: id.required(),

});

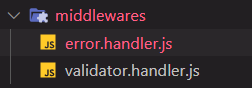
*module*.*exports* = {

  createproductSchema,

  updateproductSchema,

  getproductSchema,

};

Inside middleware folder: 

Validator.handler.js code :

*const* boom = require("@hapi/boom");

*const* validatorHandler = (*scheme*, *property*) => {

  return (*req*, *res*, *next*) => {

*const* data = req[property];

*const* { error } = scheme.validate(data);

    if (error) {

      next(boom.badRequest(error.details[0].message));

    }

    next();

  };

};

*module*.*exports* = validatorHandler;

An now we refactorize product.router.js file including this new middlewares;

*const* express = require("express");

*const* ProductServices = require("./../services/product.service");

*const* validatorHandler = require("./../middlewares/validator.handler");

*const* {

  createproductSchema,

  updateproductSchema,

  getproductSchema,

} = require("./../schemas/product.schema");

*const* router = express.Router();

*const* service = new ProductServices();

router.get("/", *async* (*req*, *res*) => {

*const* products = await service.find();

  res.json(products);

});

router.post(

  "/",

  validatorHandler(createproductSchema, "body"),

*async* (*req*, *res*) => {

*const* body = req.body;

*const* newProduct = await service.create(body);

    res.status(201).json(newProduct);

  },

);

router.patch(

  "/:id",

  validatorHandler(getproductSchema, "params"),

  validatorHandler(updateproductSchema, "body"),

*async* (*req*, *res*, *next*) => {

    try {

*const* { id } = req.params;

*const* body = req.body;

*const* product = await service.update(id, body);

      res.json(product);

    } catch (error) {

      next(error);

    }

  },

);

router.delete("/:id", *async* (*req*, *res*, *next*) => {

  try {

*const* { id } = req.params;

*const* rst = await service.delete(id);

    res.json(rst);

  } catch (error) {

    next(error);

  }

});

router.get(

  "/:id",

  validatorHandler(getproductSchema, "params"),

*async* (*req*, *res*, *next*) => {

    try {

*const* { id } = req.params;

*const* product = await service.findOne(id);

      res.json(product);

    } catch (error) {

      next(error);

    }

  },

);

router.get("/filter", (*req*, *res*) => {

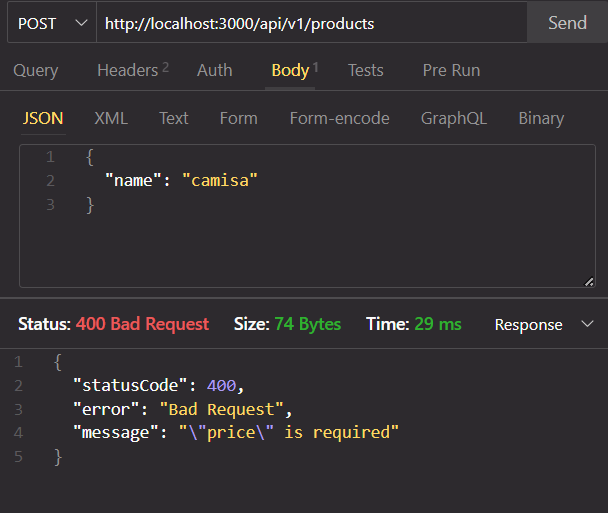
  res.send("<h1>Soy un Filter</h1>");

});

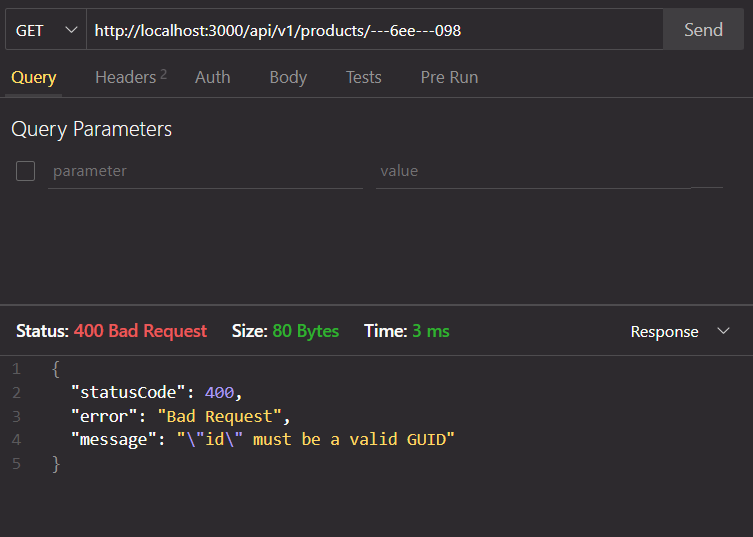
*module*.*exports* = router;

testing:

If we try to create a new product without price data:



If we try to get a product with a bad uid:



A continuación te compartiré una lista de los middlewares más populares en Express.

## CORS

Middleware para habilitar CORS (Cross-origin resource sharing) en nuestras rutas o aplicación. <http://expressjs.com/en/resources/middleware/cors.html>

## Morgan

Un logger de solicitudes HTTP para Node.js. <http://expressjs.com/en/resources/middleware/morgan.html>

## Helmet

Helmet nos ayuda a proteger nuestras aplicaciones Express configurando varios encabezados HTTP. ¡No es a prueba de balas de plata, pero puede ayudar! <https://github.com/helmetjs/helmet>

## Express Debug

Nos permite hacer debugging de nuestras aplicaciones en Express mediante el uso de un toolbar en la pagina cuando las estamos desarrollando. <https://github.com/devoidfury/express-debug>

## Express Slash

Este middleware nos permite evitar preocuparnos por escribir las rutas con o sin slash al final de ellas. <https://github.com/ericf/express-slash>

## Passport

Passport es un middleware que nos permite establecer diferentes estrategias de autenticación a nuestras aplicaciones. <https://github.com/jaredhanson/passport>

Puedes encontrar más middlewares populares en el siguiente enlace: <http://expressjs.com/en/resources/middleware.html>

Class 23 : Production.

Class 24 CORS

npm i cors To install library

npm install -g http-server

I create a new folder named frontend and create next html file:

cd . > frontend.html To create a new file using cmd

frontend.html

<!DOCTYPE *html*>

<html *lang*="en">

<head>

  <meta *charset*="UTF-8">

  <meta *http-equiv*="X-UA-Compatible" *content*="IE=edge">

  <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<body>

  <script>

    (*async* () => {

*const* response = await fetch('http://localhost:3000/api/v1/products');

*const* data = await response.json();

      console.log(data);

    })();

  </script>

</body>

</html>

Package.json

{

  "name": "frontend-my-store",

  "version": "1.0.0",

  "description": "",

  "main": "frontend.html",

  "scripts": {

    "start": "http-server -p 8080 && open http://localhost:8080/frontend/frontend.html",

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "keywords": [],

  "author": "",

  "license": "ISC"

}

And now in the API:

npm i cors To install library

Refactoring index.js file:

*const* express = require("express");

***const* cors = require("cors");**

*const* routerApi = require("./routes");

*const* {

  logErrors,

  errorHandler,

  boomErrorHandler,

} = require("./middlewares/error.handler");

*const* app = express();

*const* port = 3000;

app.use(express.json());

***const* whitelist = ["https://localhost:8080"];**

***const* corsOptions = {**

**origin: (*origin*, *callback*) => {**

**if (whitelist.includes(origin)) {**

**callback(null, true);**

**} else {**

**callback(new Error("Not allowed by CORS"));**

**}**

**},**

**};**

**app.use(cors());**

app.get("/", (*req*, *res*) => {

  res.send("Hello my server is running");

});

routerApi(app);

app.use(logErrors);

app.use(boomErrorHandler);

app.use(errorHandler);

app.listen(port, () => {

  console.log(`Server is running on port ${port}`);

});

itudi3HRK$