Gracias por la aclaración! Si estás utilizando **ES Modules** (con type: "module" en tu package.json), entonces el código debe adaptarse para usar la sintaxis de **import/export** en lugar de **require/module.exports**. Vamos a ajustar la estructura del código para que sea compatible con **ES Modules**.

**Estructura del Proyecto**

La estructura de archivos sigue siendo la misma, solo cambiaremos la sintaxis de importación/exportación.

bash

Copiar código

/api-gateway

├── /controllers

│ ├── userController.js

│ ├── productController.js

│ └── orderController.js

├── /middleware

│ └── authMiddleware.js

├── /services

│ ├── userService.js

│ ├── productService.js

│ └── orderService.js

├── /utils

│ └── logger.js

├── app.js

└── config.js

**1. Configuración de Express en app.js**

En **app.js**, importa los módulos utilizando import y exporta utilizando export default.

javascript

Copiar código

// app.js

import express from 'express';

import cors from 'cors';

import logger from './utils/logger.js';

import authMiddleware from './middleware/authMiddleware.js';

import userRoutes from './controllers/userController.js';

import productRoutes from './controllers/productController.js';

import orderRoutes from './controllers/orderController.js';

const app = express();

// Middleware global

app.use(cors());

app.use(express.json());

app.use(authMiddleware); // Middleware de autenticación

// Rutas

app.use('/users', userRoutes);

app.use('/products', productRoutes);

app.use('/orders', orderRoutes);

// Página de inicio

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

res.send('API Gateway funcionando');

});

// Manejo de errores global

app.use((err, req, res, next) => {

logger.error(err);

res.status(500).json({ error: 'Error en el servidor' });

});

export default app;

**2. Controladores (Controllers)**

En los controladores, debes importar y exportar utilizando la sintaxis de **ES Modules**.

**userController.js**

javascript

Copiar código

// controllers/userController.js

import express from 'express';

import userService from '../services/userService.js';

const router = express.Router();

// Obtener usuario por ID

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

try {

const user = await userService.getUserById(req.params.id);

res.json(user);

} catch (error) {

res.status(500).json({ error: 'No se pudo obtener el usuario' });

}

});

export default router;

**productController.js**

javascript

Copiar código

// controllers/productController.js

import express from 'express';

import productService from '../services/productService.js';

const router = express.Router();

// Obtener producto por ID

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

try {

const product = await productService.getProductById(req.params.id);

res.json(product);

} catch (error) {

res.status(500).json({ error: 'No se pudo obtener el producto' });

}

});

export default router;

**orderController.js**

javascript

Copiar código

// controllers/orderController.js

import express from 'express';

import orderService from '../services/orderService.js';

const router = express.Router();

// Crear pedido

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

try {

const order = await orderService.createOrder(req.body);

res.json(order);

} catch (error) {

res.status(500).json({ error: 'No se pudo crear el pedido' });

}

});

export default router;

**3. Servicios (Services)**

En los servicios también utilizamos **import/export**.

**userService.js**

javascript

Copiar código

// services/userService.js

import axios from 'axios';

import config from '../config.js';

// URL del microservicio de usuarios

const userServiceUrl = config.userServiceUrl;

// Obtener un usuario por ID

async function getUserById(userId) {

try {

const response = await axios.get(`${userServiceUrl}/users/${userId}`);

return response.data;

} catch (error) {

throw new Error('Error al obtener el usuario');

}

}

export default {

getUserById

};

**productService.js**

javascript

Copiar código

// services/productService.js

import axios from 'axios';

import config from '../config.js';

// URL del microservicio de productos

const productServiceUrl = config.productServiceUrl;

// Obtener un producto por ID

async function getProductById(productId) {

try {

const response = await axios.get(`${productServiceUrl}/products/${productId}`);

return response.data;

} catch (error) {

throw new Error('Error al obtener el producto');

}

}

export default {

getProductById

};

**orderService.js**

javascript

Copiar código

// services/orderService.js

import axios from 'axios';

import config from '../config.js';

// URL del microservicio de pedidos

const orderServiceUrl = config.orderServiceUrl;

// Crear un pedido

async function createOrder(orderData) {

try {

const response = await axios.post(`${orderServiceUrl}/orders`, orderData);

return response.data;

} catch (error) {

throw new Error('Error al crear el pedido');

}

}

export default {

createOrder

};

**4. Middleware de Autenticación**

El middleware también utiliza la sintaxis de **ES Modules**.

**authMiddleware.js**

javascript

Copiar código

// middleware/authMiddleware.js

import jwt from 'jsonwebtoken';

import config from '../config.js';

function authenticateJWT(req, res, next) {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return res.status(403).json({ error: 'Acceso no autorizado' });

}

jwt.verify(token, config.jwtSecret, (err, user) => {

if (err) {

return res.status(403).json({ error: 'Token inválido' });

}

req.user = user;

next();

});

}

export default authenticateJWT;

**5. Logger**

El **logger** usa la sintaxis moderna de **ES Modules**.

**logger.js**

javascript

Copiar código

// utils/logger.js

import winston from 'winston';

const logger = winston.createLogger({

level: 'info',

format: winston.format.combine(

winston.format.colorize(),

winston.format.timestamp(),

winston.format.printf(({ timestamp, level, message }) => {

return `${timestamp} [${level}]: ${message}`;

})

),

transports: [

new winston.transports.Console(),

new winston.transports.File({ filename: 'api-gateway.log' })

]

});

export default logger;

**6. Archivo de Configuración (config.js)**

El archivo de configuración también usa export default para exportar un objeto.

javascript

Copiar código

// config.js

export default {

userServiceUrl: 'http://localhost:4001',

productServiceUrl: 'http://localhost:4002',

orderServiceUrl: 'http://localhost:4003',

jwtSecret: 'your-secret-key'

};

**7. Arrancar la Aplicación**

Finalmente, en el archivo **index.js** o similar, arrancamos el servidor.

javascript

Copiar código

// index.js

import app from './app.js';

const port = 3000;

app.listen(port, () => {

console.log(`API Gateway corriendo en http://localhost:${port}`);

});

**Conclusión**

Al usar **ES Modules**, hemos adaptado toda la estructura del código para utilizar **import** y **export**. Ahora, con esta estructura modular, cada parte de tu aplicación tiene una responsabilidad clara, lo que hace que tu código sea más mantenible y escalable.

* **app.js**: Configura el servidor y las rutas.
* **Controladores**: Manejan las solicitudes HTTP.
* **Servicios**: Gestionan la interacción con los microservicios.
* **Middleware**: Realiza la autenticación de las solicitudes.
* **Logger**: Registra eventos y errores importantes.