**4. Seguridad Adicional**

* **Protección CSRF:**  
  Si tienes formularios o endpoints sensibles, puedes integrar un middleware de CSRF (por ejemplo, con csurf).
* **Validación de Entradas:**  
  Utiliza librerías como express-validator para sanitizar y validar los datos de entrada y prevenir inyecciones.
* **Registro de Logs y Auditoría:**  
  Implementa un sistema de logging (por ejemplo, con winston) para monitorear eventos y errores.

**5. Contenerización con Docker**

Crea un Dockerfile en la raíz del proyecto:

dockerfile

CopiarEditar

# Dockerfile

FROM node:18-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "app.js"]

Construye y prueba la imagen localmente:

bash

CopiarEditar

docker build -t citas-medicas .

docker run -p 3000:3000 citas-medicas

**6. Orquestación con Kubernetes**

Prepara archivos YAML para desplegar la aplicación en Kubernetes.

**a. Deployment**

yaml

CopiarEditar

# k8s/deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: citas-medicas-deployment

spec:

replicas: 2

selector:

matchLabels:

app: citas-medicas

template:

metadata:

labels:

app: citas-medicas

spec:

containers:

- name: citas-medicas

image: tu\_usuario\_docker/citas-medicas:latest

ports:

- containerPort: 3000

env:

- name: MONGO\_URL

value: "mongodb://<tu\_mongo\_host>:27017/citas\_medicas\_db"

**b. Service**

yaml

CopiarEditar

# k8s/service.yaml

apiVersion: v1

kind: Service

metadata:

name: citas-medicas-service

spec:

selector:

app: citas-medicas

ports:

- protocol: TCP

port: 80

targetPort: 3000

type: LoadBalancer

**c. Horizontal Pod Autoscaler (HPA)**

yaml

CopiarEditar

# k8s/hpa.yaml

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: citas-medicas-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: citas-medicas-deployment

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

Estos archivos permiten desplegar la aplicación en Kubernetes y configurar el escalamiento automático cuando se supere el 50% de utilización de CPU.

**7. Pruebas y Despliegue Final**

1. **Pruebas Locales:**
   * Usa herramientas como Postman para probar el registro, login, creación de citas, actualización, cancelación y consulta de citas.
   * Verifica que la autenticación JWT funcione correctamente y que los roles (paciente y médico) se apliquen en la lógica de negocio.
2. **Documentación:**
   * Accede a /documentacion para ver la documentación generada por Swagger.
3. **Publica en Git y DockerHub:**
   * Sube tu código a un repositorio Git.
   * Construye la imagen Docker y súbela a DockerHub.
4. **Despliegue en Kubernetes:**
   * Aplica los archivos YAML con kubectl apply -f k8s/.
   * Monitorea el despliegue y ajusta los parámetros según sea necesario.

Esta guía te ayudará a desarrollar el proyecto final de citas médicas paso a paso, implementando una arquitectura orientada a servicios, asegurando la seguridad de la API y preparando el despliegue en un entorno de contenedores y orquestación. Si tienes dudas en algún paso o requieres mayor detalle en alguna sección (por ejemplo, validación de entradas o integración de CSRF), puedes profundizar en esos temas o consultarme para ampliar la información.

Authmiddleware.js

const jwt = require('jsonwebtoken');

var Users = require('./model/Users');

const authenticateJWT = async (req, res, next) => {

const token = req.header('Authorization');

if (!token) {

return res.status(401).json({ error: "Acceso denegado" });

}

try {

const user = await jwt.verify(token, 'tu-palabra-secreta');

console.log(user.api\_key);

const usuario = await Users.findOne({ api\_key: user.api\_key });

if (!usuario) {

return res.status(404).json({ error: "Usuario no valido" });

}

if (usuario.saldo <= 0) {

return res.status(404).json({ error: "Saldo insuficiente" });

}

req.user = user;

next();

} catch (error) {

return res.status(400).json({ error: error.message });

}

};

module.exports = authenticateJWT;

cluster.js

const os = require('os');

const http = require('http');

//npm i cluster http-proxy

const cluster = require('cluster');

const { createProxyServer } = require('http-proxy');

const { set } = require('mongoose');

const master = require('cluster/lib/master');

// Obtener el número de CPUs disponibles en el sistema

const numCPUs = os.cpus().length;

const MAX\_PODS = 10; // Número máximo de pods

let numPods = Math.min(4, numCPUs); // Iniciamos con 4 o el número de CPUs disponibles

const PORT = 8000; // Puerto en el que el balanceador de carga estará escuchando las peticiones

// Verificar si el proceso actual es un cluster

if (cluster.isMaster) {

console.log('Servidor Maestro iniciado en el puerto:', PORT);

// Arreglo para almacenar los pods activos

const pods = [];

const podsStats = [];

// Creamos los pods

for (let i = 0; i < numPods; i++) {

const pod = cluster.fork(); // Creamos un nuevo pod

pods.push(pod); // Lo almacenamos en la lista de pods

}

//creamos un servidor para el monitoreo

const express = require('express');

const path = require('path');

const app = express();

const monitorPORT = 8080;

app.set('views engine', 'ejs');

app.set('views', path.join(\_\_dirname, 'views'));

//Rutas para estadiscticas

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

res.render('stats');

});

// Ruta para obtener los datos de las estadísticas

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

res.json(stats);

},

);

app.listen(monitorPORT, () => {

console.log('Servidor de monitoreo escuchando en el puerto:', monitorPORT);

}

);

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

const memoryUsage = process.memoryUsage().rss / 1024 / 1024; // Convertir a MB

const cpuUsage = os.loadavg()[0] / numCPUs \* 100; // Obtener la carga de la CPU

res.json({

master: {

pid: process.pid,

cpu: cpuUsage.toFixed(2),

memory: memoryUsage.toFixed(2),

podsCount: pods.length,

},

pods:Object.values(podsStats),

});

});

// Iniciar el servidor de monitoreo

// Índice para el balanceo de carga (round-robin)

let podSeleccionado = 0;

// Instancia del proxy para redirigir el tráfico a los pods

const proxy = createProxyServer();

// Creamos el servidor que actuará como balanceador de carga

const server = http.createServer((req, res) => {

if (pods.length === 0) {

res.writeHead(503, { 'Content-Type': 'text/plain' });

res.end('No hay pods disponibles');

return;

}

// Seleccionamos un pod en orden (round-robin)

const pod = pods[podSeleccionado % pods.length];

podSeleccionado++;

// Construir la URL del destino del pod

const target = 'http://localhost:' + pod.port;

console.log('Redirigiendo a:', req.url, 'al pod en el puerto:', pod.port);

// Redirigir la petición al pod seleccionado

proxy.web(req, res, { target }, (err) => {

res.writeHead(500, { 'Content-Type': 'text/plain' });

res.end('Error en el Balanceador de Carga');

console.error('Error al redirigir la petición a', target, err);

});

});

// Iniciar el servidor en el puerto definido

server.listen(PORT, () => {

console.log('Balanceador de carga escuchando en el puerto:', PORT);

});

// Manejar errores en los pods muertos

cluster.on('exit', (pod) => {

const newPod = cluster.fork();

pods[pods.indexOf(pod)] = newPod; // Reemplazar el pod muerto por uno nuevo

});

cluster.on('online', (pod) => {

pod.on('message', (message) => {

if (message.port) {

pod.port = message.port;

}

if (message.stats) {

podsStats[pod.process.pid] = { pid: pod.process.pid, cpu: message.stats.cpu, memory: message.stats.memory };

console.log('Pod', pod.process.pid, 'CPU:', message.stats.cpu, 'Memoria:', message.stats.memory);

}

});

});

// Monitorear el Servidor Master cada 5 segundos

setInterval(() => {

const memoryUsage = process.memoryUsage().rss / 1024 / 1024; // Convertir a MB

const cpuUsage = os.loadavg()[0] / numCPUs \* 100; // Obtener la carga de la CPU

console.log('Servidor maestro:', process.pid, 'CPU:', cpuUsage.toFixed(2), '%', 'Memoria:', memoryUsage.toFixed(2), 'MB');

// Si el uso de CPU es mayor al 50%, creamos un nuevo pod

if (cpuUsage > 50 && pods.length < MAX\_PODS) {

const newPod = cluster.fork();

pods.push(newPod);

numPods++;

console.log('El uso del CPU está al 50%, creando un nuevo pod en el puerto:', newPod.port);

}

}, 5000);

} else {

// Código que va a ejecutar los pods

const express = require('express');

// Importamos la conexión a la base de datos

require('./db');

// Importamos los CORS

const cors = require('cors');

// Instalamos el body-parser

const bodyParser = require('body-parser');

const helmet = require('helmet');

const path = require('path');

// Rutas

const routes = require('./rutas');

// Importar configuración de swagger

const { swaggerUi, swaggerDocs, optionsV1 } = require('./swagger');

const rateLimit = require('express-rate-limit');

const app = express();

const podPORT = 3000 + cluster.worker.id;

// Agregamos un secreto a cookie-parser

app.use(express.json());

app.use(bodyParser.urlencoded({ extended: true }));

app.use(cors());

const limiter = rateLimit({

windowMs: 15 \* 60 \* 1000, // Limitar a 15 minutos

max: 28, // Limite de solicitudes por minuto

message: 'Has excedido el límite de solicitudes, por favor espera un minuto'

});

app.use(limiter);

// Limitar el tamaño de las solicitudes

app.use(bodyParser.json({ limit: '1mb' }));

// Importamos nuestras rutas

app.use(routes);

// Página para el cambio de contraseña seguro con protección CSRF

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

const token = req.csrfToken();

// Establecer el token previamente generado en una cookie

res.cookie('csrf-token', token, { httpOnly: true });

res.render('change-password', { csrfToken: token });

});

app.post('/change-password', (req, res) => {

const token = req.body.\_csrf;

res.send('Contraseña cambiada');

const newPassword = req.body.newPassword;

if (newPassword) {

res.send('Contraseña cambiada');

} else {

res.send('Error al cambiar la contraseña');

}

});

app.use('/documentacion', swaggerUi.serve, swaggerUi.setup(swaggerDocs, optionsV1));

// Ruta principal para verificar que el pod maneja la petición

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

res.send('Pod:' + process.pid + ' escuchando en el puerto:' + podPORT);

});

setInterval(() => {

const memoryUsage = process.memoryUsage().rss / 1024 / 1024; // Convertir a MB

const cpuUsage = os.loadavg()[0] / numCPUs \* 100; // Obtener la carga de la CPU

process.send({ stats: { cpu: cpuUsage.toFixed(2), memory: memoryUsage.toFixed(2) } });

}, 5000);

// Iniciar el servidor

const server = app.listen(podPORT, () => {

console.log('Pod escuchando en el puerto:', podPORT);

process.send({ port: podPORT }); // Enviar el puerto al master

});

}

db.js

ar mongoose = require('mongoose');

var MONGO\_URL = process.env.MONGO\_URL || 'mongodb://127.0.0.1:27017/libreria\_itz\_db';

mongoose.connect(MONGO\_URL);

mongoose.connection.on('error', (err) => {

console.log('Error de conexion', err);

process.exit(1);

});

mongoose.connection.on('connected', () => {

console.log('Conexión establecida con éxito');

});

mongoose.connection.on('disconnected', () => {

console.log('Conexión finalizada');

});

//process.on('SIGINT', () => {

//mongoose.connection.close(() => {

// console.log('Conexión finalizada');

// process.exit(0);

// });

//});

process.on('SIGINT', async () => {

try {

await mongoose.connection.close();

console.log('Conexión finalizada');

process.exit(0);

} catch (err) {

console.error('Error al cerrar la conexión:', err);

process.exit(1);

}

});

module.exports = mongoose;

dbsql.js

//npm i sqlite3

const sqlite3 = require('sqlite3').verbose();

const db = new sqlite3.Database(':memory:');

db.serialize(() => {

db.run(`CREATE TABLE users(

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT UNIQUE,

password TEXT

)`);

//create a few users

db.run(`INSERT INTO users(username, password) VALUES('admin', 'admin')`);

db.run(`INSERT INTO users(username, password) VALUES('user1', 'password1')`);

db.run(`INSERT INTO users(username, password) VALUES('user2', 'password2')`);

db.run(`INSERT INTO users(username, password) VALUES('user3', 'password3')`);

console.log('Tabla de usuarios creada y usuario admin insertado');

//check size of the table

console.log('Checking the size of the table');

db.get(`SELECT COUNT(\*) as count FROM users`, (err, row) => {

console.log(row.count);

});

});

module.exports = db;

index.js

const express = require('express');

//importamos la conexion a la base de datos

require('./db');

//importamos los CORS

//const cors = require('cors');

//Instalamos el body-parser

const bodyParser = require('body-parser');

const helmet = require('helmet');

const path = require('path');

//rutas

const routes = require('./rutas');

//importar configuracion de swagger

const {swaggerUi, swaggerDocs, optionsV1} = require('./swagger');

const rateLimit = require('express-rate-limit');

const app = express();

const PORT = 3000;

//Agregamos un secreto a cookie-parser

app.use(express.json());

app.use(bodyParser.urlencoded({ extended: true }));

const cors = require('cors');

app.use(cors());

const limiter = rateLimit({

windowMs: 15 \* 60 \* 1000, //limitar a un minuto

max: 28, //limite de solicitudes por minuto

message: 'Has excedido el limite de solicitudes, porfavor espera un minuto'

});

app.use(limiter);

//limitar el tamaño de las solicitudes

app.use(bodyParser.json({limit: '5mb'}));

//const csrfProtection = csrf({ cookie: true });

app.use(routes);

//app.use(csrfProtection);

// Pagina para el cambio de contraseña seguro con protección CSRF

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

//const token = csrfProtection.create();

//const token = req.csrfToken();

//Establecer el token previamente generado en una cookie

res.cookie('csrf-token', token, {httpOnly: true});

res.render('change-password', {csrfToken: token});

});

app.post('/change-password', (req, res) => {

const token = req.body.\_csrf;

res.send('Contraseña cambiada');

const newPassword = req.body.newPassword;

if(newPassword) {

res.send('Contraseña cambiada');

} else {

res.send('Error al cambiar la contraseña');

}

});

app.use('/documentacion', swaggerUi.serve, swaggerUi.setup(swaggerDocs, optionsV1));

app.listen(PORT, () => {

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

});

Rutas.js

const express = require('express');

const librosController = require('./controllers/LibrosController');

const usersController = require('./controllers/usersController');

const authenticateJWT = require('./authMiddleware');

const router = express.Router();

// ===================== RUTAS NO PROTEGIDAS =====================

/\*\*

\* @swagger

\* /get-token:

\* post:

\* summary: Autenticación de usuario y obtención de token JWT

\* description: Permite autenticar a un usuario mediante email y api\_key y devuelve un token JWT.

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* type: object

\* properties:

\* email:

\* type: string

\* example: usuario@example.com

\* api\_key:

\* type: string

\* example: "123456789"

\* responses:

\* '200':

\* description: Autenticación exitosa

\* '404':

\* description: Usuario no encontrado o credenciales incorrectas

\* '500':

\* description: Error en el servidor

\*/

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

const { email, api\_key } = req.body;

try {

const result = await usersController.authenticate(email, api\_key);

res.json(result);

} catch (error) {

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

}

});

/\*\*

\* @swagger

\* /users:

\* post:

\* summary: Crear un nuevo usuario

\* description: Permite registrar un nuevo usuario con email y contraseña.

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* type: object

\* properties:

\* email:

\* type: string

\* example: usuario@example.com

\* password:

\* type: string

\* example: "password123"

\* responses:

\* '201':

\* description: Usuario creado con éxito

\* '400':

\* description: Datos inválidos o usuario ya existente

\* '500':

\* description: Error en el servidor

\*/

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

usersController.create(req, res);

});

// ===================== RUTAS PROTEGIDAS =====================

/\*\*

\* @swagger

\* /libros:

\* get:

\* summary: Obtener todos los libros

\* security:

\* - BearerAuth: []

\* responses:

\* '200':

\* description: Respuesta exitosa con la lista de libros

\* '500':

\* description: Error en el servidor

\*/

router.get('/libros', authenticateJWT, async (req, res) => {

librosController.list(req, res);

});

/\*\*

\* @swagger

\* /libros/{id}:

\* get:

\* summary: Obtener un libro por ID

\* security:

\* - BearerAuth: []

\* parameters:

\* - in: path

\* name: id

\* required: true

\* schema:

\* type: string

\* responses:

\* '200':

\* description: Libro encontrado

\* '404':

\* description: Libro no encontrado

\* '500':

\* description: Error en el servidor

\*/

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

librosController.show(req, res);

});

/\*\*

\* @swagger

\* /libros:

\* post:

\* summary: Crear un nuevo libro

\* security:

\* - BearerAuth: []

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* type: object

\* properties:

\* titulo:

\* type: string

\* autor:

\* type: string

\* year:

\* type: number

\* example:

\* titulo: "El principito"

\* autor: "Antoine de Saint-Exupéry"

\* year: 1943

\* responses:

\* '201':

\* description: Libro creado con éxito

\* '500':

\* description: Error en el servidor

\*/

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

librosController.create(req, res);

});

/\*\*

\* @swagger

\* /libros/{id}:

\* put:

\* summary: Actualizar un libro por ID

\* security:

\* - BearerAuth: []

\* parameters:

\* - in: path

\* name: id

\* required: true

\* schema:

\* type: string

\* - in: body

\* name: libro

\* required: true

\* description: Datos del libro a actualizar

\* schema:

\* type: object

\* properties:

\* titulo:

\* type: string

\* autor:

\* type: string

\* year:

\* type: number

\* example:

\* titulo: "Cien años de soledad"

\* autor: "Gabriel García Márquez"

\* year: 1967

\* responses:

\* '200':

\* description: Libro actualizado con éxito

\* '404':

\* description: Libro no encontrado

\* '500':

\* description: Error en el servidor

\*/

router.put('/libros/:id', authenticateJWT, async (req, res) => {

librosController.update(req, res);

});

/\*\*

\* @swagger

\* /libros/{id}:

\* delete:

\* summary: Eliminar un libro por ID

\* security:

\* - BearerAuth: []

\* parameters:

\* - in: path

\* name: id

\* required: true

\* schema:

\* type: string

\* responses:

\* '200':

\* description: Libro eliminado con éxito

\* '404':

\* description: Libro no encontrado

\* '500':

\* description: Error en el servidor

\*/

router.delete('/libros/:id', authenticateJWT, async (req, res) => {

librosController.destroy(req, res);

});

module.exports = router;

suagger

const swaggerJSDoc = require('swagger-jsdoc');

const swaggerUi = require('swagger-ui-express');

const { SwaggerTheme, SwaggerThemeNameEnum } = require('swagger-themes');

const theme = new SwaggerTheme();

const path = require('path');

//Swagger

const swaggerOptions = {

swaggerDefinition: {

openapi: '3.0.0',

info: {

title: 'Servicio REST de LIbros',

description: 'API para administrar libros',

version: '1.0.0',

termsOfService: "http://localhost:3000/terms&quot",

contact: {

name: "Edy de Santiago",

email: "edy@gmail.com",

url: "http://localhost:3000/welcome&quot"

}

},

components: {

securitySchemes: {

apiKeyAuth: {

type: 'apiKey',

name: 'Authorization',

in: 'header',

description: 'Añade tu token de seguridad en la cabecera de la solicitud'

}

},

},

security: [

{

apiKeyAuth: [] // Aplica esta configuración por defecto

}

]

},

//apis: [path.join(\_\_dirname, 'rutas.js')],

apis: [

path.join(\_\_dirname, 'services/users.js'),

path.join(\_\_dirname, 'services/libros.js')

]

};

const optionsV1 = {

explorer: true,

customCss: theme.getBuffer(SwaggerThemeNameEnum.DARK\_MONOKAI)

};

const optionsV2 = {

explorer: true,

customCss: theme.getBuffer(SwaggerThemeNameEnum.DRACULA)

};

const swaggerDocs = swaggerJSDoc(swaggerOptions);

module.exports = {

swaggerUi,

swaggerDocs,

optionsV1

}

User

var User = require('../model/Users');

const jwt = require('jsonwebtoken');

//funcion segura nosql

exports.authenticate = async function (email, api\_key) {

if (typeof email !== 'string' || typeof api\_key !== 'string') {

return {status: 500, json: {message: "Intento de NoSql Injection"}};

}

let user = await User.findOne({email, api\_key});

if(!user){

return {status: 404, json: {message: "API key no autorizada"}};

}

if(user.saldo <= 0) {

return {status: 404, json: {message: "Saldo insuficiente"}};

}

//si la autentificacion fue correcta, le generamos un token

const token = jwt.sign({email, api\_key},'tu-palabra-secreta',

{expiresIn: '1h'});

return {token, message: "El token sera valido por 1 hora"};

}

exports.create = async function (req, res) {

//para que la api funcione y reciba el request desde el body o desde el query

console.log(req.query);

let request;

if(Object.keys(req.body).length > 0) {

request = req.body;

} else if(Object.keys(req.query).length > 0) {

request = req.query;

}

//validamos campos obligatorios

if(!request.email || !request.password) {

return res.status(400).json({message: "Faltan campos obligatorios"});

}

try {//verifica si el correo electronico existe en la base de datos

const existinguser = await User.findOne({email: request.email});

if(existinguser) {

return res.status(400).json({message: "El correo electronico ya esta registrado"});

}//crear y guardar el nuevo usuario

const user = new User(request);

await user.save();

console.log(user);

return res.json({user, message: "Usuario creado correctamente"});

} catch (error) {

return res.status(500).json({message: "Error al guardar usuario", error: error.message});

}

}

exports.updateUser = async function (req, res){

const token = req.header('Authorization');

console.log('token:'+token);

const decodedToken = jwt.verify(token, 'tu-palabra-secreta');

console.log(decodedToken);

//Obtener el usuario

const user = await User.findOne({api\_key: decodedToken.api\_key});

//verificamos si el usuario existe

if(!user) {

return res.status(404).json({message: "Usuario no encontrado"});

}

//Actualizar el saldo

user.saldo = user.saldo - 1;

await user.save();

return res.json({user, message: "Peticion ha sido exitosa"});

}

Carpetas

Controller\*\* 

Model

Public

Services

views