**🔗 Guía de Integración de Microservicio**

**📋 Pre-requisitos**

Antes de empezar, asegúrate de tener:

* ✅ Tu microservicio desarrollado y funcionando
* ✅ El sistema de autenticación ejecutándose en http://localhost
* ✅ Acceso al panel de administración (admin/admin123)

**🎯 Métodos de Integración**

**Método 1: 🚀 Nginx Automático (Recomendado)**

**Método 2: 🛠️ Middleware Manual en tu código**

**Método 3: 🔄 Validación directa por API**

**🚀 MÉTODO 1: Nginx Automático (Más Simple)**

Este método es **ideal** porque no necesitas modificar tu código existente.

**Paso 1: Registrar tu microservicio en el panel**

1. Ve a http://localhost → Login con admin/admin123
2. Ve a **Microservicios** → **Nuevo Microservicio**
3. Llena el formulario:

Nombre: Mi Microservicio de Productos

Descripción: API para gestión de productos

URL: http://mi-microservicio:8000

Versión: 1.0.0

URL de Health Check: http://mi-microservicio:8000/health (opcional)

Respuesta Esperada: OK (opcional)

☑️ Requiere autenticación: Activado

Roles permitidos: admin,user,editor

**Paso 2: Agregar tu microservicio al docker-compose.yml**

# En tu docker-compose.yml, agregar:

services:

# ... servicios existentes del auth system

mi-microservicio:

build: ./mi-microservicio # o image: mi-imagen:latest

container\_name: mi\_microservicio

ports:

- "8000:8000" # puerto interno de tu app

environment:

- NODE\_ENV=production

# ... tus variables de entorno

networks:

- auth\_network

depends\_on:

- backend # para que el auth esté listo

**Paso 3: Configurar Nginx para proteger tu microservicio**

Edita auth-system/nginx/nginx.conf y busca la sección:

#----------------MICROSERVICE TEST START-----------

Reemplázala con:

#----------------MICROSERVICE PRODUCTOS START-----------

# Microservicio de Productos

location /productos/ {

# Validar automáticamente con el sistema de auth

auth\_request /auth/validate;

# Headers automáticos con info del usuario autenticado

auth\_request\_set $user\_id $upstream\_http\_x\_user\_id;

auth\_request\_set $user\_username $upstream\_http\_x\_user\_username;

auth\_request\_set $user\_email $upstream\_http\_x\_user\_email;

auth\_request\_set $user\_permissions $upstream\_http\_x\_user\_permissions;

# Pasar headers a tu microservicio

proxy\_set\_header X-User-ID $user\_id;

proxy\_set\_header X-User-Username $user\_username;

proxy\_set\_header X-User-Email $user\_email;

proxy\_set\_header X-User-Permissions $user\_permissions;

proxy\_set\_header X-Original-URI $request\_uri;

# Si no está autenticado, redirigir al login

error\_page 401 = @redirect\_login;

# Proxy hacia tu microservicio

proxy\_pass http://mi-microservicio:8000/;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

#----------------MICROSERVICE PRODUCTOS FINAL-----------

**Paso 4: Leer la información del usuario en tu microservicio**

Tu microservicio automáticamente recibirá estos headers:

// En cualquier endpoint de tu microservicio

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

// Headers automáticos del sistema de auth

const userId = req.headers['x-user-id'];

const username = req.headers['x-user-username'];

const email = req.headers['x-user-email'];

const permissions = JSON.parse(req.headers['x-user-permissions'] || '[]');

console.log('Usuario autenticado:', {

id: userId,

username: username,

email: email,

permissions: permissions

});

// Verificar permisos específicos

if (!permissions.includes('productos.read')) {

return res.status(403).json({ error: 'No tienes permisos para ver productos' });

}

// Tu lógica normal

res.json({ productos: [...] });

});

**Paso 5: Reiniciar el sistema**

# Reiniciar nginx para cargar la nueva configuración

docker-compose restart nginx

# O reiniciar todo el sistema

docker-compose down && docker-compose up -d

**Paso 6: Probar la integración**

1. Ve a http://localhost/productos/ (¡nota la barra final!)
2. Si no estás logueado → te redirige al login automáticamente
3. Después del login → accedes a tu microservicio con headers de usuario

**🛠️ MÉTODO 2: Middleware Manual**

Si prefieres más control o no puedes usar Nginx:

**Paso 1: Instalar axios en tu microservicio**

npm install axios

**Paso 2: Crear middleware de autenticación**

// middleware/auth.js en tu microservicio

const axios = require('axios');

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

try {

// Obtener token del header Authorization

const authHeader = req.headers.authorization;

if (!authHeader || !authHeader.startsWith('Bearer ')) {

return res.redirect('http://localhost/login?redirect=' + encodeURIComponent(req.originalUrl));

}

const token = authHeader.split(' ')[1];

// Validar token con el sistema de auth

const response = await axios.get('http://localhost/api/auth/validate', {

headers: {

'Authorization': `Bearer ${token}`

},

timeout: 5000

});

// Usuario válido - agregar info al request

req.user = response.data.user;

req.permissions = response.data.user.permissions || [];

next();

} catch (error) {

console.log('Error de autenticación:', error.message);

// Si es error 401, redirigir al login

if (error.response?.status === 401) {

return res.redirect('http://localhost/login?redirect=' + encodeURIComponent(req.originalUrl));

}

// Otros errores

return res.status(500).json({

error: 'Error del sistema de autenticación',

details: error.message

});

}

};

// Función helper para verificar permisos

const requirePermission = (permission) => {

return (req, res, next) => {

if (!req.permissions || !req.permissions.includes(permission)) {

return res.status(403).json({

error: 'No tienes permisos suficientes',

required: permission,

userPermissions: req.permissions

});

}

next();

};

};

module.exports = { authMiddleware, requirePermission };

**Paso 3: Usar el middleware en tus rutas**

// En tu app principal

const express = require('express');

const { authMiddleware, requirePermission } = require('./middleware/auth');

const app = express();

// Aplicar autenticación a todas las rutas protegidas

app.use('/api', authMiddleware);

// Ejemplos de uso con permisos específicos

app.get('/api/productos',

requirePermission('productos.read'),

(req, res) => {

console.log('Usuario:', req.user);

res.json({

message: `Hola ${req.user.username}!`,

productos: [...]

});

}

);

app.post('/api/productos',

requirePermission('productos.create'),

(req, res) => {

// Solo usuarios con permiso productos.create pueden crear

res.json({ message: 'Producto creado' });

}

);

// Rutas públicas (sin autenticación)

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

res.json({ status: 'OK' });

});

**🔄 MÉTODO 3: Validación Directa por API**

Para casos más específicos:

// utils/authValidator.js

const axios = require('axios');

class AuthValidator {

constructor(authSystemUrl = 'http://localhost') {

this.authUrl = authSystemUrl;

}

async validateUser(token) {

try {

const response = await axios.get(`${this.authUrl}/api/auth/validate`, {

headers: { 'Authorization': `Bearer ${token}` },

timeout: 5000

});

return {

valid: true,

user: response.data.user

};

} catch (error) {

return {

valid: false,

error: error.response?.data?.error || 'Token inválido'

};

}

}

async getUserPermissions(userId) {

try {

const response = await axios.get(`${this.authUrl}/api/users/${userId}`, {

headers: { 'Authorization': `Bearer ${this.adminToken}` }

});

return response.data.user.permissions;

} catch (error) {

return [];

}

}

}

module.exports = AuthValidator;

**🎯 Configuración Completa Paso a Paso**

**1. Docker Compose actualizado**

# tu docker-compose.yml completo

version: '3.8'

services:

# Servicios del sistema de auth (postgres, backend, frontend, nginx)

postgres:

image: postgres:15-alpine

container\_name: auth\_db

environment:

POSTGRES\_DB: auth\_system

POSTGRES\_USER: admin

POSTGRES\_PASSWORD: password123

volumes:

- postgres\_data:/var/lib/postgresql/data

ports:

- "5432:5432"

networks:

- auth\_network

backend:

build: ./auth-system/backend

container\_name: auth\_backend

environment:

DATABASE\_URL: postgresql://admin:password123@postgres:5432/auth\_system

JWT\_SECRET: change-this-super-secret-key-in-production

ports:

- "3000:3000"

depends\_on:

- postgres

networks:

- auth\_network

frontend:

build: ./auth-system/frontend

container\_name: auth\_frontend

environment:

VITE\_API\_URL: http://localhost/api

ports:

- "3001:3000"

depends\_on:

- backend

networks:

- auth\_network

nginx:

build: ./auth-system/nginx

container\_name: auth\_proxy

ports:

- "80:80"

depends\_on:

- backend

- frontend

volumes:

- ./auth-system/nginx/nginx.conf:/etc/nginx/nginx.conf:ro

networks:

- auth\_network

# TU MICROSERVICIO AQUÍ

productos-api:

build: ./mi-microservicio-productos # ruta a tu microservicio

container\_name: productos\_api

environment:

- NODE\_ENV=production

- PORT=8000

# tus variables de entorno

ports:

- "8000:8000"

networks:

- auth\_network

depends\_on:

- backend

networks:

auth\_network:

driver: bridge

volumes:

postgres\_data:

**2. Estructura de carpetas sugerida**

mi-proyecto/

├── auth-system/ # Sistema de autenticación (el que ya tienes)

├── mi-microservicio-productos/ # Tu microservicio

│ ├── Dockerfile

│ ├── package.json

│ ├── server.js

│ └── middleware/

│ └── auth.js

├── docker-compose.yml # Configuración completa

└── README.md

**3. Probar la integración**

# 1. Levantar todo el sistema

docker-compose up -d

# 2. Verificar que los servicios estén corriendo

docker-compose ps

# 3. Registrar tu microservicio en el panel

# Ve a http://localhost → Microservicios → Nuevo

# 4. Probar acceso

curl http://localhost/productos/api/test

# → Debería redirigir al login si no estás autenticado

# 5. Probar con token

curl -H "Authorization: Bearer TU\_TOKEN" http://localhost/productos/api/test

# → Debería devolver data con headers de usuario

**🔒 Permisos y Roles**

**Definir permisos para tu microservicio**

En el sistema de auth, los permisos siguen el patrón: recurso.accion

Para tu microservicio de productos:

* productos.read - Ver productos
* productos.create - Crear productos
* productos.update - Actualizar productos
* productos.delete - Eliminar productos
* productos.admin - Administración completa

**Crear roles específicos**

1. Ve al panel → **Roles** → **Nuevo Rol**
2. Crea roles como:
   * **Vendedor**: productos.read
   * **Gestor de Productos**: productos.read, productos.create, productos.update
   * **Admin de Productos**: todos los permisos de productos

**Verificar permisos en tu código**

// Verificación simple

if (!req.permissions.includes('productos.create')) {

return res.status(403).json({ error: 'Sin permisos para crear productos' });

}

// Verificación con múltiples permisos

const hasAnyPermission = (userPermissions, requiredPermissions) => {

return requiredPermissions.some(permission =>

userPermissions.includes(permission) || userPermissions.includes('\*')

);

};

if (!hasAnyPermission(req.permissions, ['productos.update', 'productos.admin'])) {

return res.status(403).json({ error: 'Sin permisos' });

}

**🚨 Troubleshooting Común**

**Problema 1: Headers no llegan**

# Verificar que nginx esté reenviando headers

docker-compose logs nginx

# Debug en tu microservicio

console.log('Headers recibidos:', req.headers);

**Problema 2: CORS errors**

// En tu microservicio, agregar CORS

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

res.header('Access-Control-Allow-Origin', 'http://localhost');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, Authorization, X-User-ID, X-User-Username');

next();

});

**Problema 3: Redirecciones infinitas**

// Asegúrate de que las rutas de health check no requieran auth

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

res.json({ status: 'OK' }); // SIN middleware de auth

});

**Problema 4: Timeout en validación**

// Aumentar timeout y manejar errores

const response = await axios.get('http://localhost/api/auth/validate', {

headers: { 'Authorization': `Bearer ${token}` },

timeout: 10000, // 10 segundos

retry: 3

});

**✅ Checklist Final**

* [ ] Microservicio registrado en el panel de auth
* [ ] Nginx configurado con la ruta de tu microservicio
* [ ] Docker Compose actualizado con tu servicio
* [ ] Middleware de autenticación implementado
* [ ] Permisos definidos y asignados a roles
* [ ] Health check funcionando sin autenticación
* [ ] Pruebas de acceso autenticado/no autenticado
* [ ] Manejo de errores implementado

**🎉 ¡Listo!**

Tu microservicio ahora está completamente integrado con el sistema de autenticación centralizado. Los usuarios solo necesitan hacer login una vez y tendrán acceso a todos los microservicios según sus permisos.

**Acceso final:**

* Panel Admin: http://localhost
* Tu microservicio: http://localhost/productos/
* API directa: Headers con info de usuario automáticamente