**🌐 Integración Universal - Cualquier Tecnología**

**✨ ¿Por qué funciona con cualquier lenguaje?**

El sistema de autenticación funciona como un **proxy inteligente** que:

1. **Intercepta las peticiones** antes de que lleguen a tu aplicación
2. **Valida la autenticación** en el sistema centralizado
3. **Agrega headers HTTP** con la info del usuario
4. **Reenvía la petición** a tu aplicación con los datos

Tu aplicación solo necesita **leer headers HTTP** - algo que puede hacer cualquier tecnología.

**🎯 Ejemplos por Tecnología**

**🟢 Node.js / Express**

// server.js

const express = require('express');

const app = express();

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

console.log('Usuario autenticado:', { userId, username, permissions });

res.json({

message: `Hola ${username}!`,

productos: ['Producto 1', 'Producto 2']

});

});

app.listen(8000);

**🐍 Python / Flask**

# app.py

from flask import Flask, request, jsonify

import json

app = Flask(\_\_name\_\_)

@app.route('/api/productos')

def get\_productos():

# Headers automáticos del sistema de auth

user\_id = request.headers.get('X-User-ID')

username = request.headers.get('X-User-Username')

permissions = json.loads(request.headers.get('X-User-Permissions', '[]'))

print(f"Usuario autenticado: {username} (ID: {user\_id})")

print(f"Permisos: {permissions}")

return jsonify({

'message': f'Hola {username}!',

'productos': ['Producto 1', 'Producto 2']

})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=8000)

**🐍 Python / Django**

# views.py

from django.http import JsonResponse

from django.views.decorators.csrf import csrf\_exempt

import json

@csrf\_exempt

def productos\_view(request):

# Headers automáticos del sistema de auth

user\_id = request.META.get('HTTP\_X\_USER\_ID')

username = request.META.get('HTTP\_X\_USER\_USERNAME')

permissions = json.loads(request.META.get('HTTP\_X\_USER\_PERMISSIONS', '[]'))

print(f"Usuario autenticado: {username} (ID: {user\_id})")

return JsonResponse({

'message': f'Hola {username}!',

'productos': ['Producto 1', 'Producto 2']

})

# urls.py

from django.urls import path

from . import views

urlpatterns = [

path('api/productos/', views.productos\_view, name='productos'),

]

**🐍 Python / FastAPI**

# main.py

from fastapi import FastAPI, Header, Request

from typing import Optional

import json

app = FastAPI()

@app.get("/api/productos")

async def get\_productos(

request: Request,

x\_user\_id: Optional[str] = Header(None),

x\_user\_username: Optional[str] = Header(None),

x\_user\_permissions: Optional[str] = Header(None)

):

# Headers automáticos del sistema de auth

permissions = json.loads(x\_user\_permissions or '[]')

print(f"Usuario autenticado: {x\_user\_username} (ID: {x\_user\_id})")

print(f"Permisos: {permissions}")

return {

"message": f"Hola {x\_user\_username}!",

"productos": ["Producto 1", "Producto 2"]

}

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=8000)

**🐘 PHP**

<?php

// index.php

header('Content-Type: application/json');

// Headers automáticos del sistema de auth

$userId = $\_SERVER['HTTP\_X\_USER\_ID'] ?? null;

$username = $\_SERVER['HTTP\_X\_USER\_USERNAME'] ?? null;

$permissions = json\_decode($\_SERVER['HTTP\_X\_USER\_PERMISSIONS'] ?? '[]', true);

echo "Usuario autenticado: $username (ID: $userId)\n";

echo "Permisos: " . print\_r($permissions, true) . "\n";

// Tu lógica aquí

$response = [

'message' => "Hola $username!",

'productos' => ['Producto 1', 'Producto 2']

];

echo json\_encode($response);

?>

**🐘 PHP / Laravel**

<?php

// routes/api.php

use Illuminate\Http\Request;

Route::get('/productos', function (Request $request) {

// Headers automáticos del sistema de auth

$userId = $request->header('X-User-ID');

$username = $request->header('X-User-Username');

$permissions = json\_decode($request->header('X-User-Permissions', '[]'), true);

Log::info("Usuario autenticado: $username (ID: $userId)");

return response()->json([

'message' => "Hola $username!",

'productos' => ['Producto 1', 'Producto 2']

]);

});

**☕ Java / Spring Boot**

// ProductosController.java

import org.springframework.web.bind.annotation.\*;

import javax.servlet.http.HttpServletRequest;

import com.fasterxml.jackson.databind.ObjectMapper;

@RestController

@RequestMapping("/api")

public class ProductosController {

@GetMapping("/productos")

public ResponseEntity<Map<String, Object>> getProductos(HttpServletRequest request) {

// Headers automáticos del sistema de auth

String userId = request.getHeader("X-User-ID");

String username = request.getHeader("X-User-Username");

String permissionsJson = request.getHeader("X-User-Permissions");

System.out.println("Usuario autenticado: " + username + " (ID: " + userId + ")");

Map<String, Object> response = new HashMap<>();

response.put("message", "Hola " + username + "!");

response.put("productos", Arrays.asList("Producto 1", "Producto 2"));

return ResponseEntity.ok(response);

}

}

**🦀 Rust / Actix-web**

// main.rs

use actix\_web::{web, App, HttpRequest, HttpResponse, HttpServer, Result};

use serde\_json::json;

async fn get\_productos(req: HttpRequest) -> Result<HttpResponse> {

// Headers automáticos del sistema de auth

let user\_id = req.headers().get("x-user-id")

.and\_then(|h| h.to\_str().ok());

let username = req.headers().get("x-user-username")

.and\_then(|h| h.to\_str().ok());

println!("Usuario autenticado: {:?} (ID: {:?})", username, user\_id);

Ok(HttpResponse::Ok().json(json!({

"message": format!("Hola {}!", username.unwrap\_or("Usuario")),

"productos": ["Producto 1", "Producto 2"]

})))

}

#[actix\_web::main]

async fn main() -> std::io::Result<()> {

HttpServer::new(|| {

App::new()

.route("/api/productos", web::get().to(get\_productos))

})

.bind("0.0.0.0:8000")?

.run()

.await

}

**🔷 C# / ASP.NET Core**

// ProductosController.cs

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/[controller]")]

public class ProductosController : ControllerBase

{

[HttpGet]

public IActionResult GetProductos()

{

// Headers automáticos del sistema de auth

var userId = Request.Headers["X-User-ID"].FirstOrDefault();

var username = Request.Headers["X-User-Username"].FirstOrDefault();

var permissions = Request.Headers["X-User-Permissions"].FirstOrDefault();

Console.WriteLine($"Usuario autenticado: {username} (ID: {userId})");

return Ok(new {

message = $"Hola {username}!",

productos = new[] { "Producto 1", "Producto 2" }

});

}

}

**🔷 Go**

// main.go

package main

import (

"encoding/json"

"fmt"

"log"

"net/http"

)

func productosHandler(w http.ResponseWriter, r \*http.Request) {

// Headers automáticos del sistema de auth

userID := r.Header.Get("X-User-ID")

username := r.Header.Get("X-User-Username")

permissions := r.Header.Get("X-User-Permissions")

fmt.Printf("Usuario autenticado: %s (ID: %s)\n", username, userID)

fmt.Printf("Permisos: %s\n", permissions)

response := map[string]interface{}{

"message": fmt.Sprintf("Hola %s!", username),

"productos": []string{"Producto 1", "Producto 2"},

}

w.Header().Set("Content-Type", "application/json")

json.NewEncoder(w).Encode(response)

}

func main() {

http.HandleFunc("/api/productos", productosHandler)

log.Println("Servidor corriendo en puerto 8000...")

log.Fatal(http.ListenAndServe(":8000", nil))

}

**💎 Ruby / Rails**

# app/controllers/productos\_controller.rb

class ProductosController < ApplicationController

skip\_before\_action :verify\_authenticity\_token

def index

# Headers automáticos del sistema de auth

user\_id = request.headers['X-User-ID']

username = request.headers['X-User-Username']

permissions = JSON.parse(request.headers['X-User-Permissions'] || '[]')

Rails.logger.info "Usuario autenticado: #{username} (ID: #{user\_id})"

Rails.logger.info "Permisos: #{permissions}"

render json: {

message: "Hola #{username}!",

productos: ['Producto 1', 'Producto 2']

}

end

end

# config/routes.rb

Rails.application.routes.draw do

get '/api/productos', to: 'productos#index'

end

**🐳 Configuración Docker para Cualquier Tecnología**

**Para Python/Flask:**

# Dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["python", "app.py"]

**Para PHP:**

# Dockerfile

FROM php:8.1-apache

COPY . /var/www/html/

EXPOSE 80

**Para Java/Spring:**

# Dockerfile

FROM openjdk:17-jdk-slim

COPY target/mi-app.jar app.jar

EXPOSE 8000

ENTRYPOINT ["java", "-jar", "/app.jar"]

**Para Go:**

# Dockerfile

FROM golang:1.19-alpine AS builder

WORKDIR /app

COPY . .

RUN go build -o main .

FROM alpine:latest

WORKDIR /root/

COPY --from=builder /app/main .

EXPOSE 8000

CMD ["./main"]

**🔧 Configuración Universal**

**1. docker-compose.yml (funciona igual para todas las tecnologías)**

services:

# ... servicios del auth system ...

mi-microservicio:

build: ./mi-microservicio # carpeta con tu app en cualquier lenguaje

container\_name: mi\_microservicio

ports:

- "8000:8000" # puerto que use tu app

environment:

# Variables específicas de tu tecnología

- ENVIRONMENT=production

networks:

- auth\_network

depends\_on:

- backend

**2. nginx.conf (igual para todas las tecnologías)**

location /mi-servicio/ {

# Validación automática

auth\_request /auth/validate;

# Headers automáticos

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

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

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

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

proxy\_set\_header X-User-Username $username;

proxy\_set\_header X-User-Permissions $permissions;

# Redirect si no autenticado

error\_page 401 = @redirect\_login;

# Proxy a tu app (sin importar el lenguaje)

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

}

**✅ Ventajas del Método Universal**

**🎯 Para Cualquier Tecnología:**

* ✅ **Sin cambios en código existente** - Solo lees headers
* ✅ **Mismo flujo de autenticación** - Login una vez, acceso a todo
* ✅ **Mismos permisos y roles** - Gestión centralizada
* ✅ **Mismo panel de administración** - Un solo lugar para todo

**🔒 Seguridad Consistente:**

* ✅ **Tokens JWT validados** antes de llegar a tu app
* ✅ **Headers seguros** - Nginx los agrega, no se pueden falsificar
* ✅ **Redirección automática** - Sin autenticación → login automático
* ✅ **Logout centralizado** - Cerrar sesión afecta todos los servicios

**🚀 Desarrollo Rápido:**

* ✅ **Plug & Play** - Registras en el panel y funciona
* ✅ **No dependencias extra** - No instalas librerías de auth
* ✅ **Testing simple** - Headers HTTP estándares
* ✅ **Debugging fácil** - Logs centralizados

**🎯 Pasos Universales**

**Para CUALQUIER tecnología:**

1. **Registrar en el panel:**
   * Ve a http://localhost → Microservicios → Nuevo
   * Llena: nombre, URL, permisos necesarios
2. **Configurar docker-compose.yml:**
3. mi-servicio:
4. build: ./mi-servicio # tu carpeta
5. ports:
6. - "8000:8000" # tu puerto
7. networks:
8. - auth\_network
9. **Configurar nginx.conf:**
10. location /mi-ruta/ {
11. auth\_request /auth/validate;
12. # ... headers automáticos
13. proxy\_pass http://mi-servicio:8000/;
14. }
15. **En tu aplicación (cualquier lenguaje):**
16. - Leer header: X-User-ID
17. - Leer header: X-User-Username
18. - Leer header: X-User-Permissions
19. - ¡Listo! Ya tienes la info del usuario
20. **Reiniciar y probar:**
21. docker-compose restart nginx
22. # Ir a http://localhost/mi-ruta/

**🌟 Ejemplo Completo: PHP Legacy**

Imagina que tienes una aplicación PHP antigua:

<?php

// mi-app-vieja/index.php

// Headers del sistema de auth (automáticos)

$user\_id = $\_SERVER['HTTP\_X\_USER\_ID'] ?? null;

$username = $\_SERVER['HTTP\_X\_USER\_USERNAME'] ?? 'Invitado';

$permissions = json\_decode($\_SERVER['HTTP\_X\_USER\_PERMISSIONS'] ?? '[]', true);

// Tu código viejo sigue igual

echo "<h1>Bienvenido $username!</h1>";

echo "<p>ID de usuario: $user\_id</p>";

// Verificar permisos

if (in\_array('productos.read', $permissions)) {

echo "<p>Puedes ver productos ✅</p>";

} else {

echo "<p>Sin permisos para productos ❌</p>";

}

// Tu lógica de negocio antigua...

?>

**Dockerfile:**

FROM php:8.1-apache

COPY . /var/www/html/

EXPOSE 80

**En docker-compose.yml:**

mi-app-php:

build: ./mi-app-vieja

ports:

- "8080:80"

networks:

- auth\_network

**En nginx.conf:**

location /legacy/ {

auth\_request /auth/validate;

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

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

proxy\_pass http://mi-app-php:80/;

}

**¡Y listo!** Tu app PHP antigua ahora tiene autenticación moderna sin tocar el código principal.

**🎉 Conclusión**

El sistema de autenticación es **100% agnóstico** de tecnología porque:

* 🌐 **Funciona a nivel HTTP** - Todos los lenguajes entienden headers
* 🔒 **Autenticación en el proxy** - Tu app solo recibe usuarios válidos
* 📦 **Headers estándares** - Misma interfaz para todo
* 🚀 **Zero configuration** - Registras y funciona