**🚀 Sistema de Autenticación Centralizado**

**Documentación Técnica ÁGIL - Versión Simplificada**

**📖 CONTEXTO DEL SOFTWARE**

**¿Qué hace este sistema?**

Un **microservicio de autenticación centralizado** que permite:

1. **Login único** para todos los microservicios del ecosistema
2. **Panel de administración** para gestionar usuarios, roles y permisos
3. **Integración súper simple** con cualquier microservicio
4. **Todo funciona con Docker Compose** - sin complejidades

**Flujo simple:**

Usuario → Login una vez → Acceso a todos los microservicios

Admin → Panel web → Gestiona todo desde una interfaz

**Problema que resuelve:**

* ❌ Múltiples logins en diferentes microservicios
* ❌ Gestión de usuarios duplicada en cada servicio
* ❌ Complejidad para implementar autenticación
* ❌ Falta de control centralizado de accesos

**Solución:**

* ✅ Single Sign-On (SSO) automático
* ✅ Panel único para gestionar todo
* ✅ Integración de microservicios con 3 líneas de código
* ✅ Escalable hasta millones de usuarios

**🎯 NECESIDADES Y REQUERIMIENTOS**

**RF-001: Sistema de Autenticación**

* Login con usuario/contraseña
* Generación de JWT tokens (Access: 15min, Refresh: 7 días)
* Logout individual o todos los dispositivos
* Renovación automática de tokens
* Sesiones concurrentes múltiples
* Usuario admin por defecto: admin/admin123

**RF-002: Gestión de Usuarios**

* CRUD completo desde panel web
* Campos: username, email, password, nombres, apellidos, cédula, teléfono
* Estados: activo/inactivo
* Soft delete (no eliminación física)
* Política de contraseñas básica (8+ chars, mayúscula, número)

**RF-003: Sistema de Roles y Permisos**

* CRUD de roles personalizados
* Permisos granulares (ej: users.create, projects.view)
* Asignación múltiple de roles por usuario
* Matriz de permisos visual

**RF-004: Gestión de Microservicios**

* Registro manual de microservicios
* Test de conectividad desde UI
* Health checks automáticos
* Activación/desactivación de servicios

**RF-005: Panel de Administración**

* Dashboard con métricas importantes
* Gestión visual de usuarios, roles, permisos
* Monitoreo de microservicios
* Interfaz responsive y minimalista

**RF-006: API Gateway**

* Validación automática de tokens
* Redirección automática al login
* Headers automáticos con info de usuario
* Integración súper simple

**💻 STACK TECNOLÓGICO SIMPLIFICADO**

Backend:

- Runtime: Node.js 18+

- Framework: Fastify 4.x (más rápido que Express)

- Language: TypeScript (type safety, mejor DX)

- ORM: Prisma (migraciones automáticas, fácil agregar campos)

- Database: PostgreSQL 15 (robusto, escalable)

- Auth: JWT + bcrypt (standard industria)

Frontend:

- Framework: React 18 + Vite (desarrollo rápido)

- Language: TypeScript (consistency con backend)

- Styling: TailwindCSS + Shadcn/ui (desarrollo ágil)

- State: React Query + Zustand (server/client state)

- Routing: React Router v6

Infrastructure:

- Containerización: Docker + Docker Compose

- API Gateway: Nginx (configuración básica)

- Development: Solo Docker Compose local

- Production: Mismo setup, escalable después

**¿Por qué este stack?**

* **Desarrollo rápido:** Todo optimizado para velocidad
* **Fácil mantenimiento:** TypeScript end-to-end
* **Escalable:** Arquitectura lista para crecer
* **Simple:** Sin over-engineering

**🏗️ ARQUITECTURA DEL SISTEMA**

**Componentes:**

Usuario → Nginx (Gateway) → Backend (API) → PostgreSQL

↓

Frontend (React)

↓

Otros Microservicios

**Flujo de autenticación:**

1. Usuario hace login → Backend valida → Genera JWT tokens
2. Frontend guarda tokens → Requests automáticos con token
3. Microservicios externos → Nginx valida → Headers automáticos
4. Token expira → Renovación automática → Usuario no se entera

**Estrategia de tokens:**

* **Access Token:** 15 minutos (performance)
* **Refresh Token:** 7 días (control)
* **Validación:** JWT + DB para revocación instantánea

**📁 ESTRUCTURA DEL PROYECTO**

auth-system/

├── 📄 docker-compose.yml # TODO el sistema en un archivo

├── 📄 .env # Variables de entorno simples

├── 📄 README.md # Cómo usar el sistema

│

├── 📂 backend/ # 🚀 API + Base de datos

│ ├── 📄 Dockerfile # Container backend

│ ├── 📄 package.json # Dependencias Node.js

│ ├── 📄 tsconfig.json # Config TypeScript básica

│ │

│ ├── 📂 prisma/ # 🗄️ Base de datos

│ │ ├── 📄 schema.prisma # Esquema completo BD

│ │ └── 📄 seed.ts # Usuario admin inicial

│ │

│ └── 📂 src/ # 💻 Código backend

│ ├── 📄 index.ts # Punto de entrada

│ ├── 📄 app.ts # Configuración Fastify

│ │

│ ├── 📂 routes/ # 🛣️ Rutas de API

│ │ ├── 📄 auth.ts # POST /login, /logout, /refresh

│ │ ├── 📄 users.ts # CRUD usuarios

│ │ ├── 📄 roles.ts # CRUD roles y permisos

│ │ └── 📄 microservices.ts # CRUD microservicios

│ │

│ ├── 📂 services/ # 🧠 Lógica de negocio

│ │ ├── 📄 auth.service.ts # Login, tokens, validación

│ │ ├── 📄 users.service.ts # Gestión usuarios

│ │ ├── 📄 roles.service.ts # Gestión roles/permisos

│ │ └── 📄 microservices.service.ts # Health checks

│ │

│ └── 📂 utils/ # 🔧 Utilidades

│ ├── 📄 jwt.ts # Manejo tokens

│ ├── 📄 password.ts # Hash contraseñas

│ └── 📄 database.ts # Cliente Prisma

│

├── 📂 frontend/ # 🎨 Panel de administración

│ ├── 📄 Dockerfile # Container frontend

│ ├── 📄 package.json # Dependencias React

│ ├── 📄 vite.config.ts # Config Vite

│ ├── 📄 tailwind.config.js # Config Tailwind

│ │

│ └── 📂 src/ # 💻 Código frontend

│ ├── 📄 App.tsx # App principal

│ ├── 📄 main.tsx # Punto de entrada React

│ │

│ ├── 📂 pages/ # 📄 Páginas principales

│ │ ├── 📄 Login.tsx # Página de login

│ │ ├── 📄 Dashboard.tsx # Dashboard principal

│ │ ├── 📄 Users.tsx # Gestión usuarios

│ │ ├── 📄 Roles.tsx # Gestión roles

│ │ └── 📄 Microservices.tsx # Gestión microservicios

│ │

│ ├── 📂 components/ # 🧩 Componentes reutilizables

│ │ ├── 📄 Layout.tsx # Layout principal

│ │ ├── 📄 Sidebar.tsx # Menú lateral

│ │ ├── 📄 UserTable.tsx # Tabla de usuarios

│ │ └── 📄 UserForm.tsx # Formulario usuarios

│ │

│ └── 📂 services/ # 🔌 Llamadas API

│ ├── 📄 api.ts # Cliente HTTP

│ ├── 📄 auth.ts # APIs de autenticación

│ ├── 📄 users.ts # APIs de usuarios

│ └── 📄 roles.ts # APIs de roles

│

├── 📂 nginx/ # 🌐 Proxy simple

│ ├── 📄 Dockerfile # Container nginx

│ └── 📄 nginx.conf # Config básica

│

└── 📂 docs/ # 📚 Documentación mínima

├── 📄 SETUP.md # Cómo levantar el sistema

├── 📄 INTEGRATION.md # Cómo integrar microservicios

└── 📄 API.md # Endpoints principales

**🗄️ ESQUEMA DE BASE DE DATOS**

// prisma/schema.prisma

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

// ===== USUARIOS =====

model User {

id Int @id @default(autoincrement())

username String @unique

email String @unique

password String // Hash con bcrypt

firstName String

lastName String

cedula String? @unique

telefono String?

isActive Boolean @default(true)

isDeleted Boolean @default(false)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

lastLogin DateTime?

// Relaciones

userRoles UserRole[]

sessions Session[]

@@map("users")

}

// ===== ROLES =====

model Role {

id Int @id @default(autoincrement())

name String @unique

description String?

permissions Json // Array simple de permisos: ["users.create", "users.edit"]

isActive Boolean @default(true)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

userRoles UserRole[]

@@map("roles")

}

// ===== RELACIÓN USUARIOS-ROLES =====

model UserRole {

id Int @id @default(autoincrement())

userId Int

roleId Int

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

role Role @relation(fields: [roleId], references: [id], onDelete: Cascade)

@@unique([userId, roleId])

@@map("user\_roles")

}

// ===== MICROSERVICIOS =====

model Microservice {

id Int @id @default(autoincrement())

name String @unique

description String?

url String

version String @default("1.0.0")

isActive Boolean @default(true)

// Health check simple

isHealthy Boolean @default(false)

lastHealthCheck DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

@@map("microservices")

}

// ===== SESIONES SIMPLES =====

model Session {

id String @id @default(uuid())

userId Int

refreshToken String @unique

expiresAt DateTime

createdAt DateTime @default(now())

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

@@map("sessions")

}

**📦 CONFIGURACIÓN DOCKER**

**docker-compose.yml:**

version: '3.8'

services:

# Base de datos

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"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U admin -d auth\_system"]

interval: 10s

timeout: 5s

retries: 5

# Backend API

backend:

build: ./backend

container\_name: auth\_backend

environment:

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

JWT\_SECRET: your-super-secret-key-change-this

JWT\_EXPIRES\_IN: 15m

JWT\_REFRESH\_EXPIRES\_IN: 7d

NODE\_ENV: development

ports:

- "3000:3000"

depends\_on:

postgres:

condition: service\_healthy

volumes:

- ./backend:/app

- /app/node\_modules

# Frontend

frontend:

build: ./frontend

container\_name: auth\_frontend

environment:

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

ports:

- "3001:3000"

depends\_on:

- backend

volumes:

- ./frontend:/app

- /app/node\_modules

# Nginx (Proxy simple)

nginx:

build: ./nginx

container\_name: auth\_proxy

ports:

- "80:80"

depends\_on:

- backend

- frontend

volumes:

- ./nginx/nginx.conf:/etc/nginx/nginx.conf

volumes:

postgres\_data:

**Variables de entorno (.env):**

# Base de datos

POSTGRES\_DB=auth\_system

POSTGRES\_USER=admin

POSTGRES\_PASSWORD=password123

# JWT

JWT\_SECRET=change-this-in-production

JWT\_EXPIRES\_IN=15m

JWT\_REFRESH\_EXPIRES\_IN=7d

# URLs

FRONTEND\_URL=http://localhost

API\_URL=http://localhost/api

**🔧 CONFIGURACIÓN NGINX**

**nginx/nginx.conf:**

events {

worker\_connections 1024;

}

http {

upstream backend {

server backend:3000;

}

upstream frontend {

server frontend:3000;

}

server {

listen 80;

# API routes → Backend

location /api/ {

proxy\_pass http://backend;

proxy\_set\_header Host $host;

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

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

}

# Validación para microservicios externos

location = /auth/validate {

internal;

proxy\_pass http://backend/api/auth/validate;

proxy\_pass\_request\_body off;

proxy\_set\_header Content-Length "";

}

# Frontend SPA

location / {

proxy\_pass http://frontend;

proxy\_set\_header Host $host;

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

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

}

}

}

**📦 DEPENDENCIAS PRINCIPALES**

**Backend (package.json):**

{

"name": "auth-backend",

"version": "1.0.0",

"scripts": {

"dev": "tsx watch src/index.ts",

"build": "tsc",

"start": "node dist/index.js",

"db:migrate": "prisma migrate deploy",

"db:seed": "tsx prisma/seed.ts"

},

"dependencies": {

"@fastify/cors": "^9.0.1",

"@fastify/jwt": "^7.2.4",

"@prisma/client": "^5.7.1",

"fastify": "^4.24.3",

"bcryptjs": "^2.4.3",

"zod": "^3.22.4"

},

"devDependencies": {

"@types/node": "^20.10.4",

"@types/bcryptjs": "^2.4.6",

"typescript": "^5.3.3",

"tsx": "^4.6.2",

"prisma": "^5.7.1"

}

}

**Frontend (package.json):**

{

"name": "auth-frontend",

"version": "1.0.0",

"scripts": {

"dev": "vite",

"build": "tsc && vite build",

"preview": "vite preview"

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0",

"react-router-dom": "^6.8.1",

"@tanstack/react-query": "^5.17.1",

"zustand": "^4.4.7",

"axios": "^1.6.2"

},

"devDependencies": {

"@types/react": "^18.2.43",

"@types/react-dom": "^18.2.17",

"@vitejs/plugin-react": "^4.2.1",

"typescript": "^5.2.2",

"vite": "^5.0.8",

"tailwindcss": "^3.4.0",

"autoprefixer": "^10.4.16",

"postcss": "^8.4.32"

}

}

**📋 PLAN DE DESARROLLO ÁGIL**

**FASE 1 (Días 1-3): Base Funcional**

* [ ] Setup Docker Compose completo
* [ ] Backend básico con Fastify + Prisma + PostgreSQL
* [ ] Endpoints básicos: login, logout, refresh
* [ ] Frontend básico con React + Vite
* [ ] Usuario admin por defecto funcionando
* [ ] Nginx proxy básico funcionando

**Entregable:** Sistema que permite login y muestra dashboard básico

**FASE 2 (Días 4-6): CRUD Usuarios**

* [ ] Endpoints CRUD usuarios completos
* [ ] Pantalla de gestión de usuarios
* [ ] Formularios crear/editar usuario
* [ ] Tabla de usuarios con paginación
* [ ] Sistema de roles básico (crear, asignar)

**Entregable:** Gestión completa de usuarios desde web

**FASE 3 (Días 7-9): Gestión Completa**

* [ ] CRUD roles y permisos completo
* [ ] Gestión de microservicios (registrar, health check)
* [ ] Panel dashboard con métricas básicas
* [ ] Nginx con validación automática para microservicios
* [ ] Sistema de permisos funcional

**Entregable:** Sistema completo de administración

**FASE 4 (Días 10-12): Integración y Documentación**

* [ ] Middleware simple para microservicios externos
* [ ] Health checks automáticos funcionando
* [ ] Documentación de integración para desarrolladores
* [ ] Testing básico (endpoints principales)
* [ ] Scripts de deployment y backup

**Entregable:** Sistema listo para producción con documentación

**🚀 COMANDOS PARA INICIAR**

**Setup inicial:**

# 1. Crear proyecto

mkdir auth-system && cd auth-system

# 2. Crear estructura

mkdir -p backend/src/routes backend/src/services backend/src/utils backend/prisma

mkdir -p frontend/src/pages frontend/src/components frontend/src/services

mkdir -p nginx docs

# 3. Copiar archivos de configuración

# (docker-compose.yml, .env, package.json, etc.)

# 4. Levantar todo el sistema

docker-compose up -d

# 5. Ejecutar migraciones

docker-compose exec backend npm run db:migrate

docker-compose exec backend npm run db:seed

**Desarrollo diario:**

# Ver logs

docker-compose logs -f

# Reiniciar un servicio

docker-compose restart backend

# Acceder a la base de datos

docker-compose exec postgres psql -U admin -d auth\_system

# Nuevas migraciones

docker-compose exec backend npx prisma migrate dev

**Acceso al sistema:**

* **Frontend:** http://localhost
* **Backend API:** http://localhost/api
* **Base de datos:** localhost:5432 (admin/password123)
* **Login admin:** admin/admin123

**🔗 INTEGRACIÓN CON MICROSERVICIOS**

**Opción 1: Middleware Simple (Recomendado)**

// En cualquier microservicio Node.js:

const axios = require('axios');

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

const token = req.headers.authorization;

if (!token) {

return res.redirect('http://localhost/login');

}

try {

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

headers: { Authorization: token }

});

req.user = response.data.user;

req.permissions = response.data.permissions;

next();

} catch (error) {

return res.redirect('http://localhost/login');

}

};

// Usar en rutas protegidas

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

**Opción 2: Nginx Automático**

# Para microservicios, agregar en nginx.conf:

location /mi-microservicio/ {

# Validar automáticamente

auth\_request /auth/validate;

# Headers automáticos

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

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

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

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

# Redirect si no autenticado

error\_page 401 = @redirect\_login;

proxy\_pass http://mi-microservicio:3000;

}

location @redirect\_login {

return 302 http://localhost/login?redirect=$request\_uri;

}

**📚 ENDPOINTS DE API PRINCIPALES**

**Autenticación:**

POST /api/auth/login

Body: { username, password }

Response: { user, accessToken, refreshToken }

POST /api/auth/refresh

Body: { refreshToken }

Response: { accessToken }

POST /api/auth/logout

Headers: Authorization: Bearer <token>

Response: { success: true }

GET /api/auth/validate

Headers: Authorization: Bearer <token>

Response: { user, permissions }

**Usuarios:**

GET /api/users # Lista usuarios

POST /api/users # Crear usuario

GET /api/users/:id # Obtener usuario

PUT /api/users/:id # Actualizar usuario

DELETE /api/users/:id # Desactivar usuario

**Roles:**

GET /api/roles # Lista roles

POST /api/roles # Crear rol

PUT /api/roles/:id # Actualizar rol

DELETE /api/roles/:id # Eliminar rol

**Microservicios:**

GET /api/microservices # Lista microservicios

POST /api/microservices # Registrar microservicio

PUT /api/microservices/:id # Actualizar microservicio

DELETE /api/microservices/:id # Eliminar microservicio

GET /api/microservices/:id/health # Health check

**✅ FUNCIONALIDADES CONFIRMADAS**

**Core del Sistema:**

* ✅ Login/logout con JWT tokens automáticos
* ✅ Renovación automática de tokens (usuario no se entera)
* ✅ Sesiones concurrentes múltiples dispositivos
* ✅ Logout individual o de todos los dispositivos
* ✅ Usuario administrador por defecto funcional

**Gestión de Usuarios:**

* ✅ CRUD completo desde panel web
* ✅ Campos: username, email, contraseña, nombres, apellidos, cédula, teléfono
* ✅ Estados activo/inactivo con soft delete
* ✅ Validación de contraseñas básica
* ✅ Búsqueda y filtrado de usuarios

**Sistema de Roles:**

* ✅ CRUD de roles personalizados
* ✅ Permisos granulares configurables
* ✅ Asignación múltiple de roles por usuario
* ✅ Matriz visual de permisos

**Gestión de Microservicios:**

* ✅ Registro manual desde panel
* ✅ Test de conectividad en tiempo real
* ✅ Health checks automáticos
* ✅ Activación/desactivación fácil

**Panel de Administración:**

* ✅ Dashboard con métricas importantes
* ✅ Interfaz responsive y minimalista
* ✅ Navegación intuitiva
* ✅ Feedback visual claro

**Integración:**

* ✅ API Gateway con Nginx
* ✅ Validación automática de tokens
* ✅ Redirecciones automáticas al login
* ✅ Headers automáticos con info usuario
* ✅ Middleware súper simple para microservicios

**Infraestructura:**

* ✅ Docker Compose completo
* ✅ Base de datos PostgreSQL configurada
* ✅ Nginx como proxy reverso
* ✅ Desarrollo y producción con mismo setup
* ✅ Escalable horizontalmente

**🎯 OBJETIVO FINAL**

**Sistema completamente funcional en máximo 2 semanas:**

1. **Semana 1:** Core funcional (login, usuarios, roles básicos)
2. **Semana 2:** Panel completo, microservicios, integración

**Al final tendrás:**

* Sistema de autenticación robusto y escalable
* Panel de administración completo y fácil de usar
* Integración súper simple con cualquier microservicio
* Todo funcionando con un solo comando: docker-compose up
* Documentación clara para desarrolladores
* Base sólida para agregar funcionalidades futuras

**Comandos finales:**

git clone <repo>

cd auth-system

docker-compose up -d

# ¡Sistema funcionando en http://localhost!

# Login: admin/admin123