**📖 Manual Técnico - Sistema de Autenticación Centralizado**

**🚀 Guía para Desarrolladores**

**🏗️ ARQUITECTURA DEL SISTEMA**

**Stack Tecnológico**

* **Backend:** Node.js + Fastify + TypeScript
* **Frontend:** React 18 + Vite + TailwindCSS
* **Base de Datos:** PostgreSQL 15
* **ORM:** Prisma
* **Autenticación:** JWT (Access + Refresh tokens)
* **Proxy:** Nginx
* **Containerización:** Docker + Docker Compose

**Flujo de Autenticación**

Usuario → Nginx (Gateway) → Backend API → PostgreSQL

↓

Frontend React

↓

Microservicios Externos

**Estructura de Tokens JWT**

// Access Token (15 minutos)

{

user: {

id: number,

username: string,

email: string,

permissions: string[]

},

type: 'access',

exp: timestamp

}

// Refresh Token (7 días)

{

user: { ... },

type: 'refresh',

exp: timestamp

}

**🔐 ACCESO Y CONFIGURACIÓN INICIAL**

**URLs del Sistema**

* **Frontend:** http://localhost (Puerto 80)
* **Backend API:** http://localhost/api (Puerto 3000, proxied)
* **Base de Datos:** localhost:5432

**Credenciales Administrativas**

# Usuario por defecto (creado en seed)

USERNAME: admin

PASSWORD: admin123

ROLES: [super\_admin]

**Variables de Entorno Importantes**

# JWT Configuration

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

JWT\_EXPIRES\_IN=15m

JWT\_REFRESH\_EXPIRES\_IN=7d

# Database

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

# Security

MAX\_LOGIN\_ATTEMPTS=5

LOCKOUT\_TIME=15m

BCRYPT\_ROUNDS=12

**📊 DASHBOARD - Métricas y Monitoreo**

**Endpoints de API**

GET /api/dashboard

**Response Structure**

{

"userStats": {

"total": number,

"active": number,

"inactive": number

},

"roleStats": {

"total": number,

"active": number,

"inactive": number

},

"microserviceStats": {

"total": number,

"active": number,

"inactive": number,

"healthy": number,

"unhealthy": number

},

"systemHealth": {

"status": "healthy" | "unhealthy",

"uptime": number,

"database": "healthy" | "unhealthy"

}

}

**Métricas Calculadas**

* **Uptime:** process.uptime() en segundos
* **Health Checks:** Queries automáticos cada 5 minutos
* **Database Status:** Test con SELECT 1 query
* **User Activity:** Basado en lastLogin timestamps

**Implementación Frontend**

const { data: dashboardData } = useQuery<DashboardData>({

queryKey: ['dashboard'],

queryFn: async () => {

const response = await api.get('/dashboard')

return response.data

},

refetchInterval: 30000, // Auto-refresh cada 30s

})

**👥 GESTIÓN DE USUARIOS**

**Modelo de Datos (Prisma Schema)**

model User {

id Int @id @default(autoincrement())

username String @unique @db.VarChar(50)

email String @unique @db.VarChar(100)

password String @db.VarChar(255) // bcrypt hash

firstName String @db.VarChar(100)

lastName String @db.VarChar(100)

cedula String? @unique @db.VarChar(20)

telefono String? @db.VarChar(20)

isActive Boolean @default(true)

isDeleted Boolean @default(false) // Soft delete

loginAttempts Int @default(0)

lockedUntil DateTime?

createdAt DateTime @default(now())

lastLogin DateTime?

userRoles UserRole[]

sessions Session[]

}

**API Endpoints**

// CRUD Operations

GET /api/users // Lista con paginación

GET /api/users/:id // Usuario específico

POST /api/users // Crear usuario

PUT /api/users/:id // Actualizar usuario

DELETE /api/users/:id // Soft delete

// Estado del usuario

PUT /api/users/:id/activate // Activar usuario

PUT /api/users/:id/deactivate // Desactivar usuario

// Gestión de contraseñas

PUT /api/users/:id/password // Cambiar contraseña

// Gestión de roles

GET /api/users/:id/roles // Roles del usuario

PUT /api/users/:id/roles // Asignar roles

**Validaciones de Backend**

// Schema de validación (Zod)

const createUserSchema = z.object({

username: z.string().min(3).max(50),

email: z.string().email().max(100),

password: z.string().min(8), // + validaciones adicionales

firstName: z.string().min(1).max(100),

lastName: z.string().min(1).max(100),

cedula: z.string().max(20).optional(),

telefono: z.string().max(20).optional(),

roleIds: z.array(z.number()).optional()

})

**Política de Contraseñas**

// utils/password.ts

export function validatePasswordPolicy(password: string) {

const errors: string[] = [];

if (password.length < 8) {

errors.push('Mínimo 8 caracteres');

}

if (!/[A-Z]/.test(password)) {

errors.push('Al menos una mayúscula');

}

if (!/[a-z]/.test(password)) {

errors.push('Al menos una minúscula');

}

if (!/\d/.test(password)) {

errors.push('Al menos un número');

}

return {

isValid: errors.length === 0,

errors

};

}

**Paginación y Filtros**

// Query parameters

interface UsersQuery {

page?: number; // Default: 1

limit?: number; // Default: 10, Max: 100

search?: string; // Busca en username, email, nombres

isActive?: boolean; // Filtro por estado

roleId?: number; // Filtro por rol específico

sortBy?: string; // Campo para ordenar

sortOrder?: 'asc' | 'desc';

}

**Frontend - React Query Implementation**

const { data: usersData, isLoading } = useQuery({

queryKey: ['users', { page: currentPage, search }],

queryFn: () => usersApi.getUsers({

page: currentPage,

limit: 10,

search: search || undefined

}),

})

**🛡️ SISTEMA DE ROLES Y PERMISOS**

**Modelo RBAC (Role-Based Access Control)**

model Role {

id Int @id @default(autoincrement())

name String @unique @db.VarChar(50)

description String?

permissions Json @default("[]") // Array de strings

isActive Boolean @default(true)

userRoles UserRole[]

}

model UserRole {

id Int @id @default(autoincrement())

userId Int

roleId Int

assignedBy Int?

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

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

@@unique([userId, roleId])

}

**Permisos Granulares**

// Permisos disponibles en el sistema

const AVAILABLE\_PERMISSIONS = [

// Gestión de usuarios

'users.create',

'users.read',

'users.update',

'users.delete',

// Gestión de roles

'roles.create',

'roles.read',

'roles.update',

'roles.delete',

// Gestión de microservicios

'microservices.create',

'microservices.read',

'microservices.update',

'microservices.delete',

// Sistema

'system.config',

'system.logs',

'dashboard.view',

// Super admin (bypass)

'\*'

]

**Middleware de Autorización**

// Verificación de permisos en rutas

export function requirePermission(permission: string) {

return async (request: FastifyRequest, reply: FastifyReply) => {

const userPermissions = request.user?.permissions || [];

// Super admin bypass

if (userPermissions.includes('\*')) {

return;

}

if (!userPermissions.includes(permission)) {

reply.code(403);

return reply.send({

error: 'Permisos insuficientes',

required: permission

});

}

};

}

// Uso en rutas

fastify.post('/api/users', {

preHandler: [fastify.authenticate, requirePermission('users.create')]

}, handler);

**Roles por Defecto**

// Roles creados en el seed

const defaultRoles = [

{

name: 'super\_admin',

permissions: ['\*'], // Todos los permisos

description: 'Acceso total al sistema'

},

{

name: 'admin',

permissions: [

'users.create', 'users.read', 'users.update',

'roles.read', 'microservices.read',

'dashboard.view'

]

},

{

name: 'user',

permissions: ['dashboard.view', 'profile.read']

}

]

**🔧 MICROSERVICIOS - Integración y Monitoreo**

**Modelo de Datos**

model Microservice {

id Int @id @default(autoincrement())

name String @unique @db.VarChar(100)

description String?

url String @db.VarChar(255)

version String @default("1.0.0")

healthCheckUrl String? @db.VarChar(255)

expectedResponse String?

requiresAuth Boolean @default(true)

allowedRoles Json @default("[]")

isActive Boolean @default(true)

isHealthy Boolean @default(false)

lastHealthCheck DateTime?

}

**Health Check Implementation**

// Servicio de health checks

export class MicroservicesService {

async performHealthCheck(id: number) {

const microservice = await prisma.microservice.findUnique({

where: { id }

});

const healthUrl = microservice.healthCheckUrl ||

`${microservice.url}/health`;

try {

const response = await axios.get(healthUrl, {

timeout: 5000,

validateStatus: (status) => status < 500

});

const isHealthy = this.validateHealthResponse(

response,

microservice.expectedResponse

);

await prisma.microservice.update({

where: { id },

data: {

isHealthy,

lastHealthCheck: new Date()

}

});

return { isHealthy, responseTime: Date.now() - startTime };

} catch (error) {

// Marcar como unhealthy

await this.markAsUnhealthy(id);

throw error;

}

}

}

**Nginx Integration - Auth Subrequest**

# nginx.conf - Validación automática

location = /auth/validate {

internal;

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

proxy\_pass\_request\_body off;

proxy\_set\_header Content-Length "";

}

# Protección de microservicio

location /protected-service/ {

auth\_request /auth/validate;

# Headers automáticos con datos del usuario

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

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

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

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

# Redirect si no autenticado

error\_page 401 = @redirect\_login;

proxy\_pass http://microservice:3000/;

}

location @redirect\_login {

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

}

**Middleware para Microservicios Node.js**

// Middleware simple para microservicios externos

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');

}

};

// Uso

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

**Health Check Scheduling**

// Programación automática de health checks

export async function scheduleHealthChecks() {

// Health check inicial

await this.performHealthCheckAll();

// Programar cada 5 minutos

setInterval(async () => {

try {

await this.performHealthCheckAll();

} catch (error) {

logger.error('Error en health check programado:', error);

}

}, 5 \* 60 \* 1000);

}

**🔐 AUTENTICACIÓN Y SEGURIDAD**

**JWT Token Flow**

// Generación de tokens

export function generateAccessToken(user: any): string {

return jwt.sign({

user: {

id: user.id,

username: user.username,

email: user.email,

permissions: user.permissions

},

type: 'access'

}, JWT\_SECRET, { expiresIn: '15m' });

}

export function generateRefreshToken(user: any): string {

return jwt.sign({

user: { id: user.id, username: user.username },

type: 'refresh'

}, JWT\_SECRET, { expiresIn: '7d' });

}

**Renovación Automática Frontend**

// Interceptor axios para renovación automática

api.interceptors.response.use(

(response) => response,

async (error) => {

const originalRequest = error.config;

if (error.response?.status === 401 && !originalRequest.\_retry) {

originalRequest.\_retry = true;

const refreshToken = useAuthStore.getState().refreshToken;

try {

const response = await axios.post('/api/auth/refresh', {

refreshToken

});

const { accessToken } = response.data;

useAuthStore.getState().setTokens(accessToken, refreshToken);

// Reintentar request original

originalRequest.headers.Authorization = `Bearer ${accessToken}`;

return api(originalRequest);

} catch (refreshError) {

useAuthStore.getState().logout();

}

}

return Promise.reject(error);

}

);

**Session Management**

model Session {

id String @id @default(uuid())

userId Int

refreshToken String @unique

userAgent String?

ipAddress String?

expiresAt DateTime

isActive Boolean @default(true)

isRevoked Boolean @default(false)

createdAt DateTime @default(now())

lastUsed DateTime @default(now())

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

}

**Rate Limiting**

// Configuración Fastify

await app.register(rateLimit, {

max: 100,

timeWindow: '1 minute',

errorResponseBuilder: (request, context) => ({

code: 429,

error: 'Too Many Requests',

message: `Rate limit exceeded, retry in ${Math.round(context.ttl / 1000)} seconds`

})

});

**Password Security**

// Hash con bcrypt

export async function hashPassword(password: string): Promise<string> {

const rounds = parseInt(process.env.BCRYPT\_ROUNDS || '12');

return await bcrypt.hash(password, rounds);

}

// Verificación

export async function verifyPassword(

password: string,

hash: string

): Promise<boolean> {

return await bcrypt.compare(password, hash);

}

**🚀 DEPLOYMENT Y CONFIGURACIÓN**

**Docker Compose Structure**

services:

postgres:

image: postgres:15-alpine

environment:

POSTGRES\_DB: auth\_system

POSTGRES\_USER: admin

POSTGRES\_PASSWORD: password123

volumes:

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

healthcheck:

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

interval: 10s

timeout: 5s

retries: 5

backend:

build: ./backend

environment:

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

JWT\_SECRET: your-secret-key

NODE\_ENV: development

depends\_on:

postgres:

condition: service\_healthy

frontend:

build: ./frontend

environment:

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

nginx:

build: ./nginx

ports:

- "80:80"

depends\_on:

- backend

- frontend

**Environment Variables**

# Producción - Cambiar estos valores

JWT\_SECRET=your-super-secure-secret-key-256-bits

DATABASE\_URL=postgresql://user:pass@host:5432/dbname

# Desarrollo

NODE\_ENV=development

LOG\_LEVEL=debug

# Seguridad

MAX\_LOGIN\_ATTEMPTS=5

LOCKOUT\_TIME=15m

BCRYPT\_ROUNDS=12

# Tokens

JWT\_EXPIRES\_IN=15m

JWT\_REFRESH\_EXPIRES\_IN=7d

**Database Migrations**

# Generar nueva migración

npx prisma migrate dev --name add\_new\_feature

# Ejecutar migraciones en producción

npx prisma migrate deploy

# Ejecutar seed

npx prisma db seed

**Comandos de Desarrollo**

# Levantar todo el sistema

docker-compose up -d

# Ver logs

docker-compose logs -f backend

docker-compose logs -f frontend

# Reiniciar servicios

docker-compose restart backend

# Ejecutar migraciones

docker-compose exec backend npx prisma migrate deploy

# Acceder a la base de datos

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

**🧪 TESTING Y DEBUGGING**

**Testing Endpoints**

# Login

curl -X POST http://localhost/api/auth/login \

-H "Content-Type: application/json" \

-d '{"username":"admin","password":"admin123"}'

# Crear usuario (con token)

curl -X POST http://localhost/api/users \

-H "Content-Type: application/json" \

-H "Authorization: Bearer YOUR\_TOKEN" \

-d '{

"username":"testuser",

"email":"test@example.com",

"password":"TestPass123",

"firstName":"Test",

"lastName":"User",

"roleIds":[3]

}'

# Health check

curl http://localhost/health

**Debugging Common Issues**

# Ver logs en tiempo real

docker-compose logs -f backend | grep ERROR

# Verificar estado de la base de datos

docker-compose exec postgres pg\_isready -U admin

# Verificar migraciones

docker-compose exec backend npx prisma migrate status

# Reset completo (CUIDADO: Borra datos)

docker-compose down -v

docker-compose up -d

**Frontend Debugging**

// Store de autenticación (Zustand)

const useAuthStore = create<AuthState>()(

persist(

(set, get) => ({

// Estado de autenticación persistente

user: null,

accessToken: null,

refreshToken: null,

isAuthenticated: false,

// Acciones

login: async (username, password) => { /\* ... \*/ },

logout: () => { /\* ... \*/ },

setTokens: (accessToken, refreshToken) => { /\* ... \*/ }

}),

{

name: 'auth-storage',

partialize: (state) => ({

user: state.user,

accessToken: state.accessToken,

refreshToken: state.refreshToken,

isAuthenticated: state.isAuthenticated,

}),

}

)

)

**🔧 EXTENSIBILIDAD Y CUSTOMIZACIÓN**

**Agregando Nuevos Permisos**

// 1. Actualizar array de permisos en RolesService

const AVAILABLE\_PERMISSIONS = [

// ... permisos existentes

'reports.create',

'reports.read',

'reports.delete'

];

// 2. Crear middleware específico

export function requireReportsAccess() {

return requirePermission('reports.read');

}

// 3. Aplicar en rutas

fastify.get('/api/reports', {

preHandler: [fastify.authenticate, requireReportsAccess()]

}, handler);

**Nuevos Campos en Usuario**

// 1. Actualizar schema de Prisma

model User {

// ... campos existentes

department String? @db.VarChar(100)

hireDate DateTime?

isManager Boolean @default(false)

}

// 2. Generar migración

// npx prisma migrate dev --name add\_user\_fields

// 3. Actualizar validaciones Zod

const createUserSchema = z.object({

// ... campos existentes

department: z.string().max(100).optional(),

hireDate: z.string().datetime().optional(),

isManager: z.boolean().optional()

});

**Custom Middleware para Microservicios**

// Middleware avanzado con caché

class AuthMiddleware {

private cache = new Map();

private cacheTimeout = 5 \* 60 \* 1000; // 5 minutos

async validate(token: string) {

// Verificar caché primero

const cached = this.cache.get(token);

if (cached && Date.now() - cached.timestamp < this.cacheTimeout) {

return cached.user;

}

// Validar con auth service

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

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

});

// Cachear resultado

this.cache.set(token, {

user: response.data.user,

timestamp: Date.now()

});

return response.data.user;

}

}

**📊 MÉTRICAS Y MONITORING**

**Health Check Endpoints**

// Sistema

GET /health

{

"status": "ok",

"timestamp": "2025-01-01T00:00:00Z",

"uptime": 3600,

"environment": "development",

"version": "1.0.0"

}

// Base de datos

GET /api/dashboard

{

"systemHealth": {

"status": "healthy",

"uptime": 3600,

"database": "healthy"

}

}

**Logging Structure**

// Logger configuración

const logger = {

error: (message: string, meta?: any) => void,

warn: (message: string, meta?: any) => void,

info: (message: string, meta?: any) => void,

debug: (message: string, meta?: any) => void,

// Logs especializados

audit: (userId: number, action: string, resource: string) => void,

request: (method: string, url: string, statusCode: number) => void

};

// Uso

logger.audit(userId, 'CREATE\_USER', 'users');

logger.request('POST', '/api/users', 201);

**🛡️ SEGURIDAD Y MEJORES PRÁCTICAS**

**Checklist de Seguridad**

**Producción**

* [ ] Cambiar JWT\_SECRET por uno seguro (256 bits)
* [ ] Usar HTTPS en todas las comunicaciones
* [ ] Configurar CORS apropiadamente
* [ ] Implementar rate limiting por IP
* [ ] Logs de auditoría habilitados
* [ ] Backup automático de base de datos
* [ ] Monitoreo de errores (Sentry, etc.)

**Desarrollo**

* [ ] Variables de entorno en .env (no committed)
* [ ] Validación en frontend Y backend
* [ ] Sanitización de inputs
* [ ] Principio de menor privilegio
* [ ] Testing de endpoints críticos

**Configuración HTTPS (Producción)**

server {

listen 443 ssl http2;

ssl\_certificate /path/to/cert.pem;

ssl\_certificate\_key /path/to/key.pem;

# Headers de seguridad

add\_header Strict-Transport-Security "max-age=31536000" always;

add\_header X-Frame-Options "SAMEORIGIN" always;

add\_header X-Content-Type-Options "nosniff" always;

add\_header Referrer-Policy "strict-origin-when-cross-origin" always;

}

**🚨 TROUBLESHOOTING TÉCNICO**

**Errores Comunes y Soluciones**

**Error 500 en creación de usuarios**

# 1. Verificar logs

docker-compose logs --tail=50 backend | grep ERROR

# 2. Problemas típicos:

# - Contraseña no cumple política

# - Usuario/email duplicado

# - Roles inexistentes

# - Conexión a BD

# 3. Debug paso a paso

curl -X POST http://localhost/api/users \

-H "Content-Type: application/json" \

-H "Authorization: Bearer TOKEN" \

-d '{"username":"test","email":"test@test.com","password":"Test123","firstName":"Test","lastName":"User"}'

**JWT Token Issues**

// Verificar token manualmente

const jwt = require('jsonwebtoken');

const token = 'YOUR\_TOKEN\_HERE';

const secret = 'YOUR\_JWT\_SECRET';

try {

const decoded = jwt.verify(token, secret);

console.log('Token válido:', decoded);

} catch (error) {

console.log('Token inválido:', error.message);

}

**Database Connection Issues**

# Verificar conexión

docker-compose exec postgres psql -U admin -d auth\_system -c "SELECT NOW();"

# Verificar logs de Postgres

docker-compose logs postgres

# Reset de BD (CUIDADO)

docker-compose down

docker volume rm auth-system\_postgres\_data

docker-compose up -d

**Frontend Build Issues**

# Limpiar y rebuild

docker-compose down

docker-compose build --no-cache frontend

docker-compose up -d

# Verificar variables de entorno

docker-compose exec frontend env | grep VITE

**📚 REFERENCIAS Y RECURSOS**

**Documentación Técnica**

* **Fastify:** https://www.fastify.io/docs/
* **Prisma:** https://www.prisma.io/docs/
* **React Query:** https://tanstack.com/query/latest
* **TailwindCSS:** https://tailwindcss.com/docs
* **JWT:** https://jwt.io/introduction

**Arquitectura de Referencia**

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Frontend │ │ Backend API │ │ PostgreSQL │

│ React + Vite │◄──►│ Fastify + TS │◄──►│ + Prisma ORM │

└─────────────────┘ └─────────────────┘ └─────────────────┘

▲ ▲ ▲

│ │ │

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Nginx Proxy │ │ JWT Tokens │ │ Docker Comp │

│ + Rate Limit │ │ + Sessions │ │ + Volumes │

└─────────────────┘ └─────────────────┘ └─────────────────┘

**Comandos de Referencia Rápida**

# Sistema

docker-compose up -d # Iniciar todo

docker-compose logs -f backend # Ver logs

docker-compose restart backend # Reiniciar servicio

# Base de datos

npx prisma migrate dev # Nueva migración

npx prisma studio # GUI de BD

npx prisma db seed # Ejecutar seed

# Testing

curl http://localhost/health # Health check

curl http://localhost/api # Info de API

**¡Con esta documentación técnica puedes entender, mantener y extender el sistema de autenticación de manera profesional!** 🚀