# 📝 Comentarios y README del Código - PepsiCo Fleet Management

**Proyecto:** PepsiCo Fleet Management System

**Versión:** 1.0.0

**Fecha:** Octubre 15, 2024

**Equipo:** Joaquín Marín & Benjamin Vilches

---

## 📋 Índice

1. [README Principal](#readme-principal)
2. [README del Backend](#readme-del-backend)
3. [README del Frontend](#readme-del-frontend)
4. [Comentarios en el Código](#comentarios-en-el-código)
5. [Estándares de Documentación](#estándares-de-documentación)
6. [Herramientas de Documentación](#herramientas-de-documentación)

---

## 1. README Principal

### 1.1 Estructura del README.md

# 🚛 Plataforma de Gestión de Ingreso de Vehículos - PepsiCo Chile  
  
## 🎯 Estado del Proyecto  
- ✅ COMPLETADO - Fase de Construcción y Documentación Técnica  
- 📅 Fecha: Octubre 15, 2024  
- 🏆 Progreso: 100% de funcionalidades implementadas  
- 📚 Documentación: ~1,600 páginas de documentación técnica  
  
## 🚀 Funcionalidades Implementadas  
- ✅ Sistema de Autenticación con JWT y RBAC  
- ✅ 6 Dashboards Personalizados por rol de usuario  
- ✅ 80+ Endpoints REST completamente documentados  
- ✅ 20 Tablas de Base de Datos con integridad referencial  
- ✅ Sistema de Notificaciones automáticas  
- ✅ Control de Inventario con alertas de stock  
- ✅ Gestión de Órdenes de Trabajo con seguimiento completo  
- ✅ Sistema de Auditoría para todas las acciones  
- ✅ Interfaz Responsive con diseño corporativo PepsiCo  
  
## 📋 Tabla de Contenidos  
- [Descripción](#descripción)  
- [Stack Tecnológico](#stack-tecnológico)  
- [Estructura del Proyecto](#estructura-del-proyecto)  
- [Requisitos Previos](#requisitos-previos)  
- [Instalación](#instalación)  
- [Configuración](#configuración)  
- [Ejecución](#ejecución)  
- [Base de Datos](#base-de-datos)  
- [Deployment](#deployment)  
- [Equipo](#equipo)

### 1.2 Secciones Clave

#### Descripción del Proyecto

## 📝 Descripción  
  
Plataforma web que digitaliza y automatiza la gestión de ingresos de vehículos al taller, reemplazando el proceso manual actual basado en planillas Excel y WhatsApp. El sistema permite:  
  
- ✅ Registro de ingreso/salida de vehículos con captura de fotos  
- ✅ Gestión de órdenes de trabajo (OT)  
- ✅ Control de inventario de repuestos  
- ✅ Asignación de mecánicos y seguimiento de estados  
- ✅ Sistema de notificaciones automáticas  
- ✅ Generación de reportes de productividad  
- ✅ Control de acceso basado en roles (RBAC)  
- ✅ Gestión de 10 perfiles de usuario diferenciados

#### Stack Tecnológico

## 🛠️ Stack Tecnológico  
  
### Frontend  
- \*\*React 18.3+\*\* - Librería UI  
- \*\*TypeScript 5.5+\*\* - Lenguaje tipado  
- \*\*Vite 5.4+\*\* - Build tool  
- \*\*TailwindCSS 3.4+\*\* - Estilos  
- \*\*shadcn/ui\*\* - Componentes UI  
- \*\*React Router 6.26+\*\* - Navegación  
- \*\*TanStack Query 5.56+\*\* - Estado servidor  
- \*\*Zustand 4.5+\*\* - Estado global  
- \*\*React Hook Form + Zod\*\* - Formularios y validación  
- \*\*Recharts 2.12+\*\* - Gráficos  
  
### Backend  
- \*\*Node.js 20 LTS\*\* - Runtime  
- \*\*Express.js 4.19+\*\* - Framework web  
- \*\*TypeScript 5.5+\*\* - Lenguaje tipado  
- \*\*Prisma ORM 5.20+\*\* - ORM  
- \*\*PostgreSQL 15+\*\* - Base de datos  
- \*\*JWT\*\* - Autenticación  
- \*\*bcrypt\*\* - Hash de contraseñas  
- \*\*Winston\*\* - Logging  
- \*\*Nodemailer\*\* - Email  
- \*\*pdfkit + exceljs\*\* - Generación de reportes

---

## 2. README del Backend

### 2.1 Estructura del backend/README.md

# 🚀 Backend API - Plataforma de Gestión de Flota  
  
API REST desarrollada con Node.js, Express y TypeScript para la gestión de ingresos de vehículos.  
  
## 📊 Estadísticas del Backend  
  
- \*\*📁 Archivos:\*\* 41 archivos TypeScript  
- \*\*📝 Líneas de Código:\*\* ~4,600 líneas  
- \*\*🔗 Endpoints:\*\* 80+ endpoints REST  
- \*\*🗄️ Tablas:\*\* 20 tablas en PostgreSQL  
- \*\*🔒 Seguridad:\*\* JWT + RBAC + Auditoría  
- \*\*📚 Documentación:\*\* 100% documentado  
  
## 🏗️ Arquitectura

Backend/

├── src/

│ ├── controllers/ # 10 controladores (1,200 líneas)

│ ├── services/ # 9 servicios (2,000 líneas)

│ ├── routes/ # 11 archivos de rutas (600 líneas)

│ ├── middlewares/ # 5 middlewares (400 líneas)

│ ├── utils/ # 4 utilidades (300 líneas)

│ └── config/ # 2 configuraciones (100 líneas)

├── prisma/

│ ├── schema.prisma # Esquema de BD (440 líneas)

│ └── seed.ts # Datos de prueba (280 líneas)

└── tests/ # Pruebas (próxima fase)

### 2.2 Endpoints Documentados

## 📡 Endpoints  
  
### Autenticación  
- `POST /api/auth/login` - Iniciar sesión  
- `POST /api/auth/refresh` - Renovar token  
- `POST /api/auth/logout` - Cerrar sesión  
- `GET /api/auth/me` - Obtener usuario actual  
- `POST /api/auth/change-password` - Cambiar contraseña  
  
### Usuarios  
- `GET /api/users` - Listar usuarios  
- `GET /api/users/:id` - Obtener usuario  
- `POST /api/users` - Crear usuario  
- `PUT /api/users/:id` - Actualizar usuario  
- `DELETE /api/users/:id` - Eliminar usuario  
- `POST /api/users/:id/restore` - Restaurar usuario  
  
### Vehículos  
- `GET /api/vehicles` - Listar vehículos  
- `GET /api/vehicles/:id` - Obtener vehículo  
- `GET /api/vehicles/plate/:licensePlate` - Buscar por patente  
- `POST /api/vehicles` - Crear vehículo  
- `PUT /api/vehicles/:id` - Actualizar vehículo  
- `DELETE /api/vehicles/:id` - Eliminar vehículo  
  
### Ingresos de Vehículos  
- `GET /api/vehicle-entries` - Listar ingresos  
- `GET /api/vehicle-entries/:id` - Obtener ingreso  
- `POST /api/vehicle-entries` - Registrar ingreso  
- `PUT /api/vehicle-entries/:id` - Actualizar ingreso  
- `POST /api/vehicle-entries/:id/exit` - Registrar salida  
- `PUT /api/vehicle-entries/:id/keys` - Actualizar control de llaves  
  
### Órdenes de Trabajo  
- `GET /api/work-orders` - Listar OT  
- `GET /api/work-orders/:id` - Obtener OT  
- `POST /api/work-orders` - Crear OT  
- `PUT /api/work-orders/:id` - Actualizar OT  
- `POST /api/work-orders/:id/status` - Cambiar estado  
- `POST /api/work-orders/:id/pause` - Pausar trabajo  
- `POST /api/work-orders/:id/resume` - Reanudar trabajo  
- `POST /api/work-orders/:id/photos` - Subir fotos  
  
### Inventario  
- `GET /api/spare-parts` - Listar repuestos  
- `GET /api/spare-parts/low-stock` - Stock bajo  
- `POST /api/spare-parts` - Crear repuesto  
- `PUT /api/spare-parts/:id` - Actualizar repuesto  
- `POST /api/spare-parts/:id/adjust-stock` - Ajustar stock  
- `POST /api/spare-parts/request` - Solicitar repuesto  
- `POST /api/spare-parts/deliver/:id` - Entregar repuesto  
  
### Dashboard  
- `GET /api/dashboard/stats` - Estadísticas generales  
- `GET /api/dashboard/stats/:period` - Estadísticas por período  
- `GET /api/dashboard/mechanics-performance` - Rendimiento mecánicos  
- `GET /api/dashboard/activity` - Actividad reciente

---

## 3. README del Frontend

### 3.1 Estructura del frontend/README.md

# 🎨 Frontend - Plataforma de Gestión de Flota  
  
Aplicación web desarrollada con React, TypeScript y Vite.  
  
## 📊 Estadísticas del Frontend  
  
- \*\*📁 Archivos:\*\* 25 archivos TypeScript/TSX  
- \*\*📝 Líneas de Código:\*\* ~2,850 líneas  
- \*\*🎨 Componentes:\*\* 10+ componentes reutilizables  
- \*\*📱 Páginas:\*\* 8 páginas principales  
- \*\*🎯 Dashboards:\*\* 6 dashboards personalizados por rol  
- \*\*📚 Documentación:\*\* 100% documentado  
  
## 🏗️ Arquitectura

Frontend/

├── src/

│ ├── components/ # 10 componentes (800 líneas)

│ ├── pages/ # 8 páginas (1,500 líneas)

│ ├── services/ # 5 servicios (400 líneas)

│ ├── store/ # 1 store Zustand (100 líneas)

│ ├── hooks/ # Custom hooks (50 líneas)

│ └── utils/ # Utilidades (50 líneas)

├── public/ # Assets estáticos

└── dist/ # Build de producción

### 3.2 Dashboards Implementados

## 🎯 Dashboards por Rol  
  
### 1. Dashboard de Administrador  
- Vista completa del sistema  
- Estadísticas globales  
- Gestión de usuarios  
- Configuración del sistema  
  
### 2. Dashboard de Guardia  
- Control de ingreso/salida vehicular  
- Registro rápido de ingresos  
- Búsqueda de vehículos  
- Control de llaves  
  
### 3. Dashboard de Recepcionista  
- Vehículos sin orden asignada  
- Crear órdenes de trabajo  
- Asignar mecánicos  
- Vehículos listos para salida  
  
### 4. Dashboard de Mecánico  
- Mis órdenes asignadas  
- Trabajos en progreso  
- Solicitar repuestos  
- Historial de trabajos  
  
### 5. Dashboard de Jefe de Taller  
- Supervisión completa del taller  
- Estado de todas las órdenes  
- Rendimiento de mecánicos  
- Asignación de órdenes  
  
### 6. Dashboard de Encargado de Inventario  
- Alertas de stock crítico  
- Solicitudes de repuestos  
- Movimientos recientes  
- Entregar repuestos

---

## 4. Comentarios en el Código

### 4.1 Estándares de Comentarios

#### JSDoc para Funciones

/\*\*  
 \* Obtiene un usuario por ID con sus relaciones  
 \* @param id - ID único del usuario  
 \* @param includeRelations - Si incluir relaciones (rol, taller)  
 \* @returns Usuario encontrado con sus relaciones  
 \* @throws Error si usuario no existe  
 \* @example  
 \* ```typescript  
 \* const user = await getUserById('123e4567-e89b-12d3-a456-426614174000', true)  
 \* console.log(user.role.name) // 'Administrador'  
 \* ```  
 \*/  
async function getUserById(id: string, includeRelations = false): Promise<User> {  
 // Implementación  
}

#### Comentarios de Clase

/\*\*  
 \* Servicio para gestión de usuarios del sistema  
 \*   
 \* Proporciona operaciones CRUD para usuarios, incluyendo:  
 \* - Creación y actualización de usuarios  
 \* - Asignación de roles y talleres  
 \* - Validación de permisos  
 \* - Auditoría de cambios  
 \*   
 \* @class UserService  
 \* @version 1.0.0  
 \* @since 2024-10-01  
 \*/  
export class UserService {  
 // Implementación  
}

#### Comentarios de Sección

// ============================================================================  
// AUTHENTICATION MIDDLEWARE  
// ============================================================================  
  
/\*\*  
 \* Middleware de autenticación JWT  
 \*   
 \* Verifica que el token JWT sea válido y no haya expirado.  
 \* Extrae información del usuario del token y la agrega al request.  
 \*   
 \* @param req - Request object  
 \* @param res - Response object   
 \* @param next - Next middleware function  
 \*/  
export async function authenticate(req: Request, res: Response, next: NextFunction) {  
 // Implementación  
}  
  
// ============================================================================  
// AUTHORIZATION MIDDLEWARE   
// ============================================================================  
  
/\*\*  
 \* Middleware de autorización RBAC  
 \*   
 \* Verifica que el usuario tenga permisos para acceder al recurso.  
 \* Utiliza el sistema de roles y permisos configurado.  
 \*   
 \* @param resource - Recurso a acceder (ej: 'users', 'vehicles')  
 \* @param action - Acción a realizar (ej: 'read', 'create', 'update')  
 \*/  
export function authorize(resource: string, action: string) {  
 // Implementación  
}

### 4.2 Comentarios de Configuración

#### Variables de Entorno

// ============================================================================  
// ENVIRONMENT CONFIGURATION  
// ============================================================================  
  
/\*\*  
 \* Configuración de variables de entorno  
 \*   
 \* Todas las variables de entorno están documentadas en env.example.txt  
 \* Para desarrollo local, copiar env.example.txt a .env y configurar valores  
 \*   
 \* Variables críticas:  
 \* - DATABASE\_URL: URL de conexión a PostgreSQL  
 \* - JWT\_SECRET: Secreto para firmar tokens JWT  
 \* - FRONTEND\_URL: URL del frontend para CORS  
 \*/  
  
const config = {  
 // Base de datos  
 database: {  
 url: process.env.DATABASE\_URL || 'postgresql://localhost:5432/pepsico\_fleet',  
 // Configuración de conexión  
 connectionLimit: parseInt(process.env.DB\_CONNECTION\_LIMIT || '10'),  
 acquireTimeout: parseInt(process.env.DB\_ACQUIRE\_TIMEOUT || '60000'),  
 },  
   
 // JWT  
 jwt: {  
 secret: process.env.JWT\_SECRET || 'dev-secret-change-in-production',  
 expiresIn: process.env.JWT\_EXPIRES\_IN || '15m',  
 refreshExpiresIn: process.env.JWT\_REFRESH\_EXPIRES\_IN || '7d',  
 },  
   
 // Servidor  
 server: {  
 port: parseInt(process.env.PORT || '3000'),  
 nodeEnv: process.env.NODE\_ENV || 'development',  
 },  
   
 // CORS  
 cors: {  
 origin: process.env.FRONTEND\_URL || 'http://localhost:5173',  
 credentials: true,  
 }  
}

### 4.3 Comentarios de Lógica de Negocio

#### Validaciones

/\*\*  
 \* Valida RUT chileno usando algoritmo de dígito verificador  
 \*   
 \* @param rut - RUT a validar (con o sin puntos y guión)  
 \* @returns true si el RUT es válido, false en caso contrario  
 \*   
 \* @example  
 \* ```typescript  
 \* validateRUT('12345678-9') // true  
 \* validateRUT('12345678-K') // true   
 \* validateRUT('12345678-0') // false  
 \* ```  
 \*/  
export function validateRUT(rut: string): boolean {  
 // Limpiar RUT (remover puntos y guión)  
 const cleanRUT = rut.replace(/\./g, '').replace(/-/g, '')  
   
 // Separar cuerpo y dígito verificador  
 const body = cleanRUT.slice(0, -1)  
 const dv = cleanRUT.slice(-1).toUpperCase()  
   
 // Validar formato básico  
 if (!/^\d+$/.test(body) || !/^[0-9K]$/.test(dv)) {  
 return false  
 }  
   
 // Calcular dígito verificador esperado  
 let sum = 0  
 let multiplier = 2  
   
 for (let i = body.length - 1; i >= 0; i--) {  
 sum += parseInt(body[i]) \* multiplier  
 multiplier = multiplier === 7 ? 2 : multiplier + 1  
 }  
   
 const expectedDV = 11 - (sum % 11)  
 const calculatedDV = expectedDV === 11 ? '0' : expectedDV === 10 ? 'K' : expectedDV.toString()  
   
 return dv === calculatedDV  
}

#### Transacciones

/\*\*  
 \* Crea un ingreso de vehículo con control de llaves  
 \*   
 \* Esta operación es atómica - si falla cualquier paso, se revierte todo.  
 \* Pasos de la transacción:  
 \* 1. Crear registro de ingreso  
 \* 2. Actualizar estado del vehículo  
 \* 3. Crear control de llaves (si aplica)  
 \* 4. Enviar notificaciones  
 \*   
 \* @param data - Datos del ingreso  
 \* @returns Ingreso creado con relaciones  
 \* @throws Error si falla la transacción  
 \*/  
async function createVehicleEntry(data: CreateVehicleEntryData): Promise<VehicleEntry> {  
 return await prisma.$transaction(async (tx) => {  
 // 1. Crear ingreso  
 const entry = await tx.vehicleEntry.create({  
 data: {  
 entryCode: generateEntryCode(),  
 vehicleId: data.vehicleId,  
 workshopId: data.workshopId,  
 driverRut: data.driverRut,  
 driverName: data.driverName,  
 driverPhone: data.driverPhone,  
 entryDate: new Date(),  
 entryKm: data.entryKm,  
 fuelLevel: data.fuelLevel,  
 hasKeys: data.hasKeys,  
 observations: data.observations,  
 photos: data.photos,  
 status: 'ingresado',  
 createdById: data.createdById,  
 },  
 include: {  
 vehicle: true,  
 workshop: true,  
 createdBy: true,  
 }  
 })  
   
 // 2. Actualizar estado del vehículo  
 await tx.vehicle.update({  
 where: { id: data.vehicleId },  
 data: { status: 'in\_maintenance' }  
 })  
   
 // 3. Crear control de llaves si aplica  
 if (data.hasKeys) {  
 await tx.keyControl.create({  
 data: {  
 entryId: entry.id,  
 keyLocation: data.keyLocation || 'Recepción',  
 observations: data.keyObservations,  
 }  
 })  
 }  
   
 // 4. Enviar notificaciones (no crítico para la transacción)  
 try {  
 await notificationService.sendEntryNotification(entry)  
 } catch (error) {  
 // Log error pero no fallar la transacción  
 logger.warn('Failed to send entry notification', { error, entryId: entry.id })  
 }  
   
 return entry  
 })  
}

---

## 5. Estándares de Documentación

### 5.1 Estructura de Archivos

#### README por Módulo

# 📁 [Nombre del Módulo]  
  
## 📋 Descripción  
Breve descripción del módulo y su propósito.  
  
## 🏗️ Arquitectura  
Diagrama o descripción de la arquitectura del módulo.  
  
## 📡 API Reference  
Documentación de endpoints si aplica.  
  
## 🧪 Testing  
Cómo probar el módulo.  
  
## 📝 Ejemplos  
Ejemplos de uso del módulo.

#### Comentarios en Archivos

/\*\*  
 \* @fileoverview [Descripción del archivo]  
 \* @author [Nombre del autor]  
 \* @version [Versión]  
 \* @since [Fecha de creación]  
 \*/  
  
// ============================================================================  
// IMPORTS  
// ============================================================================  
  
// Node modules  
import express from 'express'  
import bcrypt from 'bcryptjs'  
  
// Internal modules  
import prisma from '../config/database'  
import { validateRUT } from '../utils/validation'  
  
// Types  
import type { Request, Response } from 'express'  
import type { User, CreateUserData } from '../types'  
  
// ============================================================================  
// CONSTANTS  
// ============================================================================  
  
const SALT\_ROUNDS = 10  
const MAX\_LOGIN\_ATTEMPTS = 5  
  
// ============================================================================  
// INTERFACES  
// ============================================================================  
  
interface LoginRequest {  
 email: string  
 password: string  
}  
  
interface LoginResponse {  
 accessToken: string  
 refreshToken: string  
 user: User  
}  
  
// ============================================================================  
// FUNCTIONS  
// ============================================================================  
  
/\*\*  
 \* Autentica un usuario con email y contraseña  
 \* @param credentials - Credenciales de login  
 \* @returns Tokens y datos del usuario  
 \*/  
async function login(credentials: LoginRequest): Promise<LoginResponse> {  
 // Implementación  
}  
  
// ============================================================================  
// EXPORTS  
// ============================================================================  
  
export { login }  
export type { LoginRequest, LoginResponse }

### 5.2 Documentación de API

#### Endpoint Documentation

/\*\*  
 \* @swagger  
 \* /api/auth/login:  
 \* post:  
 \* summary: Iniciar sesión  
 \* description: Autentica un usuario y retorna tokens JWT  
 \* tags: [Authentication]  
 \* requestBody:  
 \* required: true  
 \* content:  
 \* application/json:  
 \* schema:  
 \* type: object  
 \* required: [email, password]  
 \* properties:  
 \* email:  
 \* type: string  
 \* format: email  
 \* example: "admin@pepsico.cl"  
 \* password:  
 \* type: string  
 \* minLength: 8  
 \* example: "password123"  
 \* responses:  
 \* 200:  
 \* description: Login exitoso  
 \* content:  
 \* application/json:  
 \* schema:  
 \* type: object  
 \* properties:  
 \* success:  
 \* type: boolean  
 \* example: true  
 \* data:  
 \* type: object  
 \* properties:  
 \* accessToken:  
 \* type: string  
 \* refreshToken:  
 \* type: string  
 \* user:  
 \* $ref: '#/components/schemas/User'  
 \* 401:  
 \* description: Credenciales inválidas  
 \* content:  
 \* application/json:  
 \* schema:  
 \* $ref: '#/components/schemas/Error'  
 \*/

---

## 6. Herramientas de Documentación

### 6.1 Generación Automática

#### JSDoc

# Instalar JSDoc  
npm install -g jsdoc  
  
# Generar documentación  
jsdoc -c jsdoc.conf.json src/  
  
# Configuración jsdoc.conf.json  
{  
 "source": {  
 "include": ["./src/"],  
 "includePattern": "\\.(js|ts)$",  
 "excludePattern": "(node\_modules/|tests/)"  
 },  
 "opts": {  
 "destination": "./docs/api/",  
 "recurse": true  
 },  
 "plugins": ["plugins/markdown"],  
 "templates": {  
 "cleverLinks": false,  
 "monospaceLinks": false  
 }  
}

#### TypeDoc

# Instalar TypeDoc  
npm install -g typedoc  
  
# Generar documentación TypeScript  
typedoc --out docs/api src/ --theme default  
  
# Con configuración  
typedoc --config typedoc.json

### 6.2 Documentación en Código

#### Comentarios de TODO

// TODO: Implementar cache de usuarios para mejorar performance  
// FIXME: Corregir validación de email en edge cases  
// HACK: Solución temporal hasta implementar Redis  
// NOTE: Este endpoint será deprecado en v2.0  
// WARNING: No usar en producción sin configurar HTTPS

#### Comentarios de Performance

/\*\*  
 \* Obtiene estadísticas del dashboard  
 \*   
 \* ⚠️ PERFORMANCE: Esta query puede ser lenta con muchos datos.  
 \* Considerar implementar cache Redis para datos que no cambian frecuentemente.  
 \*   
 \* @param period - Período de estadísticas  
 \* @returns Estadísticas del dashboard  
 \*/  
async function getDashboardStats(period: string): Promise<DashboardStats> {  
 // Implementación optimizada con índices  
}

#### Comentarios de Seguridad

/\*\*  
 \* Valida permisos de usuario para acceder a recurso  
 \*   
 \* 🔒 SECURITY: Esta función es crítica para la seguridad del sistema.  
 \* Siempre validar permisos antes de permitir acceso a recursos sensibles.  
 \*   
 \* @param userId - ID del usuario  
 \* @param resource - Recurso a acceder  
 \* @param action - Acción a realizar  
 \* @returns true si tiene permisos, false en caso contrario  
 \*/  
async function validatePermissions(userId: string, resource: string, action: string): Promise<boolean> {  
 // Implementación de validación de permisos  
}

---

## 7. Checklist de Documentación

### 7.1 Para Cada Archivo

* [ ] Header con información del archivo
* [ ] Comentarios JSDoc en funciones públicas
* [ ] Comentarios inline para lógica compleja
* [ ] Ejemplos de uso cuando sea necesario
* [ ] Documentación de parámetros y retornos
* [ ] Manejo de errores documentado

### 7.2 Para Cada Módulo

* [ ] README.md con descripción del módulo
* [ ] Ejemplos de uso
* [ ] API reference si aplica
* [ ] Guía de testing
* [ ] Troubleshooting común

### 7.3 Para el Proyecto

* [ ] README principal actualizado
* [ ] Documentación de instalación
* [ ] Guía de desarrollo
* [ ] Documentación de deployment
* [ ] Changelog actualizado

---

## 8. Ejemplos de Comentarios Mejorados

### 8.1 Antes (Comentarios Básicos)

// Función para crear usuario  
function createUser(data) {  
 // Validar datos  
 if (!data.email) {  
 throw new Error('Email required')  
 }  
   
 // Crear usuario  
 const user = prisma.user.create({ data })  
 return user  
}

### 8.2 Después (Comentarios Profesionales)

/\*\*  
 \* Crea un nuevo usuario en el sistema  
 \*   
 \* Valida los datos de entrada, verifica que el email no esté en uso,  
 \* hashea la contraseña y crea el usuario con el rol especificado.  
 \*   
 \* @param data - Datos del usuario a crear  
 \* @param data.email - Email del usuario (único)  
 \* @param data.password - Contraseña en texto plano  
 \* @param data.roleId - ID del rol a asignar  
 \* @param data.workshopId - ID del taller (opcional)  
 \* @returns Usuario creado sin contraseña  
 \* @throws ValidationError si los datos son inválidos  
 \* @throws ConflictError si el email ya existe  
 \*   
 \* @example  
 \* ```typescript  
 \* const user = await createUser({  
 \* email: 'mecanico@pepsico.cl',  
 \* password: 'password123',  
 \* roleId: 'mechanic-role-id',  
 \* workshopId: 'workshop-1'  
 \* })  
 \* console.log(user.email) // 'mecanico@pepsico.cl'  
 \* ```  
 \*/  
async function createUser(data: CreateUserData): Promise<Omit<User, 'password'>> {  
 // Validar email único  
 const existingUser = await prisma.user.findUnique({  
 where: { email: data.email }  
 })  
   
 if (existingUser) {  
 throw new ConflictError('El email ya está en uso')  
 }  
   
 // Hashear contraseña  
 const hashedPassword = await bcrypt.hash(data.password, SALT\_ROUNDS)  
   
 // Crear usuario  
 const user = await prisma.user.create({  
 data: {  
 ...data,  
 password: hashedPassword,  
 isActive: true,  
 createdAt: new Date(),  
 updatedAt: new Date()  
 },  
 select: {  
 id: true,  
 email: true,  
 firstName: true,  
 lastName: true,  
 role: true,  
 workshop: true,  
 isActive: true,  
 createdAt: true,  
 updatedAt: true  
 // Excluir password del resultado  
 }  
 })  
   
 // Log de auditoría  
 await auditLogService.log({  
 action: 'user\_created',  
 resource: 'users',  
 resourceId: user.id,  
 details: { email: user.email, role: user.role.name }  
 })  
   
 return user  
}

---

**Última actualización:** Octubre 15, 2024

**Versión:** 1.0.0

**Mantenido por:** Joaquín Marín & Benjamin Vilches