

## **Persistencia de Datos**

### **Arquitectura de Base de Datos**

El sistema utiliza PostgreSQL con una estructura relacional bien definida que incluye tablas para gestión de usuarios con roles y permisos, control de proyectos con diferentes estados, inventario de materiales y movimientos de bodega. La base de datos implementa tipos ENUM para campos controlados como estados de proyectos y tipos de movimiento, asegurando consistencia en los datos.

### **Autenticación y Autorización**

Cuando un usuario se registra, su contraseña se hashea con bcrypt antes de almacenarse, mientras el sistema crea relaciones con roles y permisos mediante tablas de unión. Al iniciar sesión, se verifica el hash y se genera un token JWT que encapsula toda la información de autorización del usuario.

### **Gestión de Proyectos**

Al crear un proyecto, se validan las relaciones con clientes y se aplican constraints de presupuesto no negativo. Cuando un proyecto cambia a estado "Finalizado" o "Cancelado", un trigger automáticamente establece la fecha fin a la fecha actual, mientras que para otros estados garantiza que esta fecha sea nula.

### **Control de Inventario**

Los movimientos de bodega siguen un flujo estricto: las entradas aumentan el stock con cantidades positivas y las salidas lo disminuyen con valores negativos, siempre verificando stock disponible antes de permitir salidas. El sistema calcula dinámicamente el stock disponible restando los materiales reservados para proyectos del total en bodega.

### **Asignación de Materiales a Proyectos**

Los materiales se asignan a proyectos mediante una relación muchos-a-muchos que rastrea cantidades ofertadas, en obra y reservadas. Un trigger bloquea cualquier modificación en proyectos finalizados o cancelados, mientras que las transacciones aseguran la consistencia cuando se actualizan múltiples cantidades simultáneamente.

### **Sistema de Reportes**

Cada reporte de progreso se almacena con su avance, actividades y problemas, permitiendo el seguimiento histórico. Las fotos se gestionan mediante una tabla separada que soporta tanto URLs externas como archivos subidos, manteniendo la relación con el reporte correspondiente.

### **Mecanismos de Integridad**

El sistema emplea transacciones de base de datos para operaciones complejas que involucran múltiples tablas, como la creación de proyectos con sus materiales asignados.

Las foreign keys garantizan la integridad referencial, mientras que los constraints CHECK validan reglas de negocio como fechas coherentes y cantidades válidas. Los índices optimizan las consultas frecuentes de reportes y dashboard.

### **Estrategia de Consultas**

Para reportes y dashboards, se utilizan consultas con Common Table Expressions (CTEs) que calculan métricas complejas como popularidad de materiales y distribución de proyectos por servicio. Las vistas implícitas mediante subconsultas permiten obtener datos agregados sin almacenamiento redundante, manteniendo los datos siempre actualizados.

Esta arquitectura de persistencia asegura que todos los flujos de negocio mantengan consistencia, performance y confiabilidad, soportando las operaciones críticas de la empresa mientras previene estados inválidos mediante validaciones a múltiples niveles.

### **Persistencia de Datos: Estrategia Frontend-Backend**

El frontend maneja la persistencia mediante una arquitectura en capas donde React gestiona el estado local mientras delega la persistencia permanente al backend a través de APIs REST. Esta estrategia sirve para mantener una experiencia de usuario fluida con actualizaciones optimistas y cache inteligente, resolviendo el problema de la latencia de red y la consistencia de datos. Es importante porque permite respuestas inmediatas en la interfaz mientras se sincroniza en segundo plano con la fuente de verdad, beneficiando a los usuarios con una aplicación responsive y a los desarrolladores con un manejo simplificado de estados de sincronización.