Documento Técnico: Sistema "Cirujano de Sintetizadores"

Proyecto: cirujano-front
URL: www.cirujanodesintetizadores.cl
Repositorio: https://github.com/CristobalZurita/cirujano-front
Fecha de análisis: Enero 2026
Versión del documento: 1.0

---

1. Resumen Ejecutivo

Cirujano de Sintetizadores es una aplicación web full-stack diseñada para la gestión integral de un taller de reparación de instrumentos musicales electrónicos (sintetizadores, drum machines, teclados). El sistema combina una landing page pública con un panel administrativo completo, integrando funcionalidades de gestión de inventario, diagnósticos asistidos por IA, cotizaciones automatizadas, y seguimiento de reparaciones.

Alcance del sistema

- **Frontend público:** Landing page responsive con secciones institucionales, catálogo de servicios, formulario de contacto
- **Panel administrativo:** Gestión completa de inventario, usuarios, clientes, reparaciones y diagnósticos
- **Módulo de IA:** Análisis de imágenes para detección de fallas y generación automática de cotizaciones
- **Backend API:** Sistema RESTful completo con autenticación, auditoría y gestión de datos

---

2. Arquitectura General del Sistema

2.1 Stack Tecnológico

Frontend

- **Framework:** Vue 3 (Composition API)
- **Build Tool:** Vite
- **UI Framework:** Bootstrap 5
- **Estilos:** SCSS con sistema de theming customizado
- **State Management:** Pinia (stores modulares)
- **Routing:** Vue Router
- **Testing:** Vitest + Jest
- **Comunicación:** Axios (encapsulado en composables)

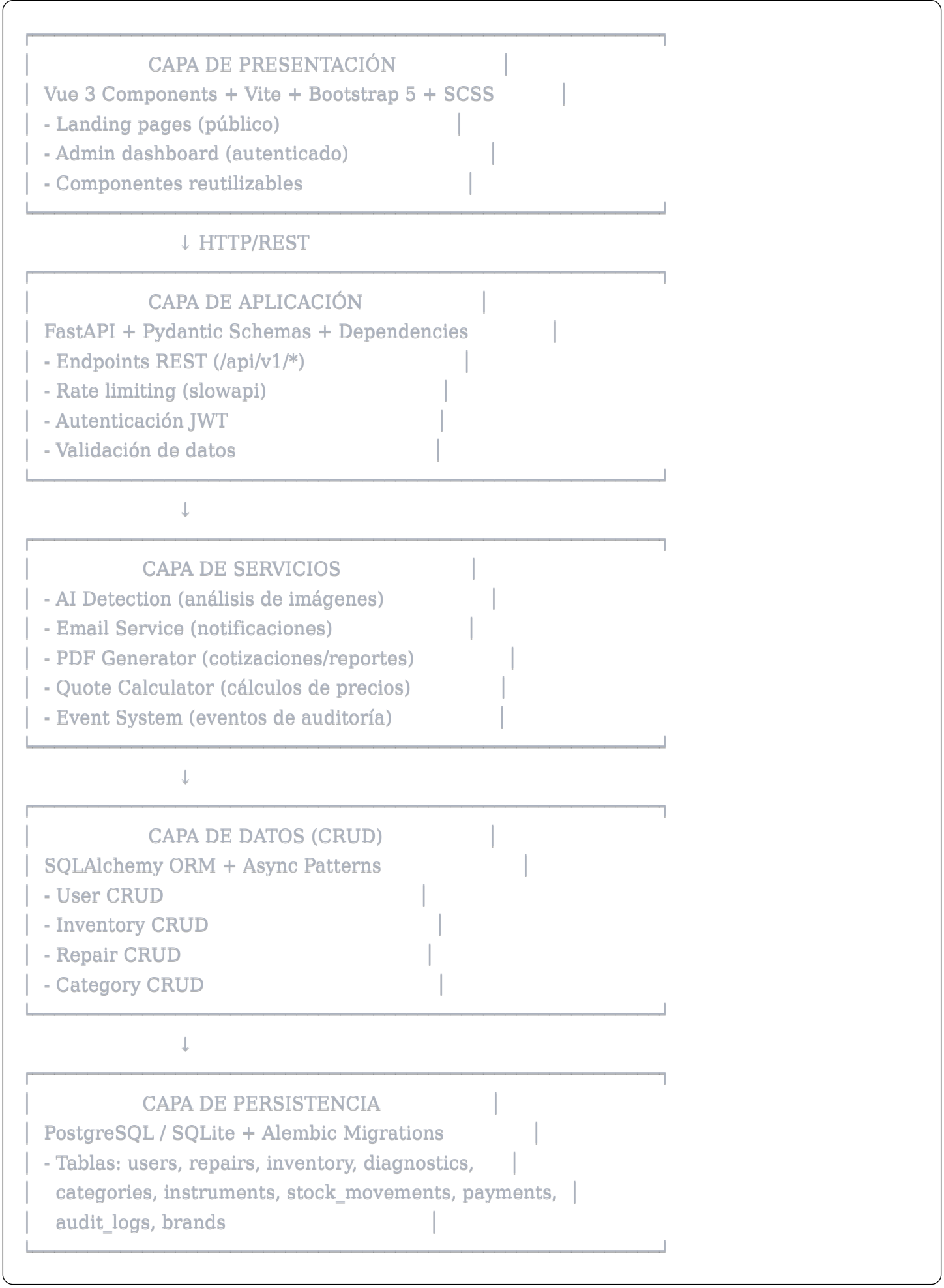
Backend

- **Framework:** FastAPI (Python 3.11+)
- **ORM:** SQLAlchemy 2.0
- **Base de datos:** PostgreSQL (producción) / SQLite (desarrollo)
- **Autenticación:** JWT con python-jose
- **Validación:** Pydantic 2.5+
- **Async DB:** asyncpg
- **Workers:** Celery + Redis
- **Testing:** pytest + pytest-asyncio
- **Logging:** python-json-logger

Servicios externos

- **Email:** EmailJS (frontend) + servicio SMTP (backend)
- **IA:** Servicio de análisis de imágenes (ai_detector.py)
- **Storage:** Sistema de archivos local para uploads

2.2 Arquitectura de Capas



---

3. Funcionalidades Implementadas

3.1 Módulo Frontend Público

Secciones principales (`src/vue/content/sections/`)

- **HeroSection:** Página de inicio con llamado a la acción
- **AboutSection:** Información institucional del taller
- **ServicesSection:** Catálogo de servicios ofrecidos
- **HistorySection:** Historia y trayectoria
- **PortfolioSection:** Galería de proyectos/trabajos
- **TeamSection:** Equipo de trabajo
- **ReviewsSection:** Testimonios de clientes
- **FaqSection:** Preguntas frecuentes
- **ContactSection:** Formulario de contacto integrado con EmailJS

Componentes reutilizables (`src/vue/components/`)

- **Navigation/Navbar:** Sistema de navegación sticky responsive
- **Footer:** Footer con múltiples columnas de información
- **Loaders:** Spinners y estados de carga
- **Widgets:** Alertas, badges, botones, tabs de filtrado
- **Layout:** Sistema de composición de páginas (PageWrapper, PageSection, etc.)

3.2 Módulo de Autenticación

Ubicación: `src/vue/components/auth/`

Componentes:

- `LoginForm.vue`: Formulario de inicio de sesión
- `RegisterForm.vue`: Registro de nuevos usuarios
- `PasswordReset.vue`: Recuperación de contraseña
- `AccountDelete.vue`: Eliminación de cuenta

Backend: `backend/app/api/v1/endpoints/auth.py`

- Registro de usuarios con hash bcrypt
- Login con generación de JWT
- Refresh tokens
- Endpoints de recuperación de contraseña
- Rate limiting en endpoints críticos

3.3 Panel Administrativo

Ubicación: `src/vue/content/pages/admin/`

Módulos implementados:

1. **AdminDashboard.vue**: Panel principal con estadísticas
   - Componente: `StatsCards.vue` (métricas clave)
   - Componente: `InventoryAlerts.vue` (alertas de stock bajo)
2. **InventoryPage.vue**: Gestión de inventario
   - `InventoryTable.vue`: Tabla con filtros y paginación
   - `InventoryForm.vue`: Formulario CRUD de items
   - Tracking de movimientos de stock (`StockMovements.vue`)
   - Store: `src/stores/inventory.js`
   - Backend: `backend/app/models/inventory.py`
3. **CategoriesPage.vue**: Gestión de categorías
   - `CategoryManager.vue`: Vista principal
   - `CategoryList.vue`: Listado con acciones
   - `CategoryForm.vue`: Formulario CRUD
   - Backend: `backend/app/models/category.py`
4. **ClientsPage.vue**: Gestión de clientes
   - `ClientList.vue`: Listado de clientes
   - `ClientDetail.vue`: Vista detallada de cliente
   - Backend: Usuario con role "client"
5. **RepairsAdminPage.vue**: Gestión de reparaciones
   - `RepairManager.vue`: Vista de gestión
   - `RepairForm.vue`: Creación/edición de reparaciones
   - `RepairStatusEditor.vue`: Actualización de estados
   - `RepairsList.vue`: Listado con filtros
   - Backend: `backend/app/models/repair.py`
6. **StatsPage.vue**: Estadísticas y reportes
   - Backend: `backend/app/api/v1/endpoints/stats.py`

3.4 Módulo de Diagnósticos con IA

Ubicación: `src/vue/components/ai/`

Componentes clave:

- `FaultDetector.vue`: Detector de fallas en imágenes
- `ImageUploader.vue`: Subida y preview de imágenes
- `FaultMarker.vue`: Marcadores visuales de fallas detectadas
- `AIAnalysisResult.vue`: Presentación de resultados de análisis
- `QuoteGenerator.vue`: Generación automática de cotizaciones

Backend: `backend/app/services/ai_detector.py`

- Análisis de imágenes de instrumentos
- Detección de componentes dañados
- Generación de diagnósticos preliminares

Flujo de trabajo:

1. Usuario sube imagen del instrumento
2. Sistema analiza imagen con IA
3. Se detectan fallas visuales
4. Se genera diagnóstico preliminar
5. Se calcula cotización automática
6. Se crea registro en base de datos

3.5 Sistema de Cotizaciones

Ubicación: `src/vue/components/quotation/`

- `InstrumentSelector.vue`: Selector de instrumento del catálogo
- `QuotationResult.vue`: Vista de cotización generada
- `DisclaimerModal.vue`: Términos y condiciones

Backend: `backend/app/services/quote_calculator.py`

- Cálculo de precios base por categoría
- Factores de ajuste según marca/modelo
- Descuentos y promociones
- Generación de PDF (`pdf_generator.py`)

3.6 Gestión de Reparaciones

Frontend: `src/vue/components/dashboard/`

- `DashboardPanel.vue`: Panel de usuario
- `RepairCard.vue`: Tarjeta individual de reparación
- `RepairTimeline.vue`: Timeline de estados
- `RepairsList.vue`: Listado de reparaciones del usuario
- `StatusBadge.vue`: Badges visuales de estado

Estados de reparación:

- Recibido
- En diagnóstico
- Esperando repuestos
- En reparación
- Control de calidad
- Finalizado
- Entregado

Backend: `backend/app/models/repair.py`

- Modelo con campos: cliente, instrumento, falla reportada, diagnóstico, costo, fechas, estado
- Relaciones: usuario (cliente), instrumento, pagos
- Auditoría automática de cambios

3.7 Gestión de Instrumentos

Frontend:

- `src/vue/components/admin/InstrumentForm.vue`
- `src/vue/components/admin/InstrumentList.vue`

Backend: `backend/app/models/instrument.py`

- Catálogo de instrumentos soportados
- Relación con marcas (`brand.py`)
- Datos técnicos y de servicio

Data: `src/assets/data/instruments.json`

- Base de datos JSON con catálogo de instrumentos
- Marcas: Korg, Roland, Akai, Access, etc.
- Modelos detallados con especificaciones

3.8 Sistema de Usuarios y Roles

Backend: `backend/app/models/user.py`

Roles implementados:

- `admin`: Acceso completo al sistema
- `tecnico`: Gestión de reparaciones y diagnósticos
- `client`: Vista de sus propias reparaciones

Funcionalidades:

- Registro con validación de email
- Hash de contraseñas con bcrypt
- JWT con expiración configurable
- Endpoints de gestión de perfil
- Sistema de permisos por endpoint

3.9 Sistema de Auditoría

Backend: `backend/app/models/audit.py`

Features:

- Logging automático de todas las operaciones CRUD
- Registro de cambios con before/after
- Tracking de usuario, timestamp, acción
- Event system para hooks personalizados (`services/event_system.py`)

---

4. Estructura de Datos

4.1 Modelos principales (Backend)

User

python

- id: UUID
- email: str (unique)
- username: str
- hashed_password: str
- full_name: str
- role: str (admin/tecnico/client)
- is_active: bool
- created_at: datetime
- updated_at: datetime

Repair

python

- id: UUID
- user_id: FK(User)
- instrument_id: FK(Instrument)
- reported_fault: str
- diagnosis: str
- status: str
- estimated_cost: Decimal
- final_cost: Decimal
- received_date: datetime
- estimated_delivery: datetime
- actual_delivery: datetime
- notes: str

Inventory

python

- id: UUID
- part_name: str
- category_id: FK(Category)
- quantity: int
- min_quantity: int
- unit_price: Decimal
- supplier: str
- location: str

Diagnostic

```python
- id: UUID
- repair_id: FK(Repair)
- image_url: str
- ai_analysis: JSON
- faults_detected: JSON
- confidence_score: float
- created_at: datetime
```

Payment

```python
- id: UUID
- repair_id: FK(Repair)
- amount: Decimal
- payment_method: str
- status: str
- transaction_id: str
- payment_date: datetime
```

4.2 Relaciones clave

```
User (1) ──→ (N) Repair
Repair (1) ──→ (N) Diagnostic
Repair (1) ──→ (N) Payment
Instrument (1) ──→ (N) Repair
Category (1) ──→ (N) Inventory
Inventory (1) ──→ (N) StockMovement
Brand (1) ──→ (N) Instrument
```

---

5. Composables y Lógica Reutilizable (Frontend)

Ubicación: `src/composables/`

- `useAuth.js`: Gestión de autenticación y sesión
- `useApi.js`: Cliente HTTP base con interceptores
- `useInventory.js`: Lógica de inventario
- `useRepairs.js`: Gestión de reparaciones
- `useDiagnostics.js`: Diagnósticos e IA
- `useCategories.js`: Categorías y taxonomía
- `useInstruments.js`: Catálogo de instrumentos
- `useQuotation.js`: Sistema de cotizaciones
- `useUsers.js`: Gestión de usuarios
- `useStockMovements.js`: Movimientos de inventario
- `scheduler.js`: Lógica de agendamiento
- `emails.js`: Integración con EmailJS
- `settings.js`: Configuración global
- `strings.js`: Internacionalización y textos
- `utils.js`: Utilidades generales

---

6. Sistema de Estilos

Ubicación: `src/scss/`

Arquitectura SCSS modular:

- `style.scss`: Entrada principal
- `_variables.scss`: Variables globales (colores, espaciados, breakpoints)
- `_brand.scss`: Identidad visual corporativa
- `_typography.scss`: Sistema tipográfico (fuentes: Cervo Neue Con, Steelfish)
- `_theming.scss`: Sistema de temas (claro/oscuro)
- `_layout.scss`: Grid y estructura base
- `_mixins.scss`: Mixins reutilizables

Fuentes customizadas:

- Cervo Neue Con (múltiples pesos)
- Steelfish RG

Ubicación: `public/fonts/`

---

7. Testing

Frontend

Ubicación: `src/components/prototypes/__tests__/`

- `InventoryCard.spec.js`
- `InventoryForm.spec.js`
- Framework: Vitest + Vue Test Utils

Backend

Ubicación: `backend/tests/`

Tests implementados:

- `test_audit_logging.py`: Sistema de auditoría
- `test_audit_hooks.py`: Event hooks
- `test_config.py`: Configuración
- `test_import_endpoints.py`: Importación de datos
- `test_payments_endpoints.py`: Pagos
- `test_payments_concurrency.py`: Concurrencia en pagos
- `test_ratelimit.py`: Rate limiting
- `test_security_scan.py`: Escaneo de seguridad
- `test_uploads.py`: Subida de archivos
- `test_items_api.py`: API de items

Framework: pytest + pytest-asyncio

---

8. Seguridad

Implementaciones de seguridad:

1. **Autenticación:**
    - JWT con HS256
    - Refresh tokens
    - Expiración configurable
    - Rate limiting en login
2. **Autorización:**
    - Decoradores de permisos por rol
    - Verificación a nivel de endpoint
    - Ownership validation
3. **Input validation:**
    - Pydantic schemas en todos los endpoints
    - Sanitización de HTML
    - Validación de tipos
4. **Rate limiting:**
    - slowapi con backend Redis
    - Límites por IP y por usuario
    - Configuración: `backend/app/core/ratelimit.py`
5. **Password security:**
    - Bcrypt hashing
    - Política de contraseñas seguras
    - No se almacenan contraseñas en texto plano
6. **Upload security:**
    - Validación de tipo MIME
    - Límite de tamaño de archivo
    - Sanitización de nombres
    - Escaneo con python-magic
7. **CORS:**
    - Configuración restrictiva
    - Whitelist de orígenes
8. **SQL Injection:**
    - Protección nativa de SQLAlchemy
    - Prepared statements
    - No queries en raw SQL

9. Configuración y Deployment

Variables de entorno (Backend)

Archivo: `.env` (no versionado)

```bash
# Database
DATABASE_URL=postgresql://user:pass@localhost/cirujano

# Security
SECRET_KEY=your-secret-key
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30

# Email
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email
SMTP_PASSWORD=your-password

# Redis
REDIS_URL=redis://localhost:6379

# Environment
ENVIRONMENT=development
DEBUG=True
```

Scripts de inicialización

Ubicación: `backend/scripts/`

- `init_db_and_seed.py`: Crea tablas e inserta datos iniciales
- `create_admin.py`: Crea usuario administrador
- `promote_to_admin.py`: Promueve usuario a admin

Migraciones

Herramienta: Alembic

Comandos:

```bash
alembic revision --autogenerate -m "descripción"
alembic upgrade head
alembic downgrade -1
```

GitHub Actions

Ubicación: `.github/workflows/`

Posible CI/CD:

- Build y test automático
- Deploy a GitHub Pages (frontend)
- Deploy a servidor (backend)

---

10. Qué Hace el Código

10.1 Frontend

El frontend es una **Single Page Application (SPA)** que proporciona:

1. **Landing page pública** con información institucional, servicios, portfolio y contacto
2. **Sistema de autenticación** con login, registro y recuperación de contraseña
3. **Dashboard de usuario** donde los clientes pueden ver sus reparaciones activas
4. **Panel administrativo** completo para gestionar inventario, categorías, clientes, reparaciones y estadísticas
5. **Módulo de diagnóstico con IA** que permite subir imágenes y obtener análisis automático
6. **Sistema de cotizaciones** que genera presupuestos basados en el instrumento y fallas detectadas

10.2 Backend

El backend es una **API RESTful** que:

1. **Gestiona autenticación y autorización** con JWT y roles
2. **Expone endpoints CRUD** para todos los modelos (usuarios, reparaciones, inventario, etc.)
3. **Procesa imágenes con IA** para detección de fallas
4. **Genera cotizaciones automáticas** basadas en reglas de negocio
5. **Envía notificaciones por email** en eventos clave
6. **Registra auditoría completa** de todas las operaciones
7. **Maneja pagos** y tracking de transacciones
8. **Genera reportes en PDF** de cotizaciones y facturas

---

11. Cómo lo Hace

11.1 Patrón de arquitectura

Frontend: Arquitectura basada en componentes con Composition API

- Separación de concerns: presentación (components) vs lógica (composables) vs estado (stores)
- Comunicación unidireccional de datos
- Componentes pequeños y reutilizables
- Stores de Pinia para estado global

Backend: Arquitectura en capas

- **Capa de presentación:** Endpoints FastAPI con validación Pydantic
- **Capa de negocio:** Services para lógica compleja
- **Capa de datos:** CRUD con SQLAlchemy ORM
- **Capa de persistencia:** PostgreSQL/SQLite

11.2 Flujo típico de una operación

Ejemplo: **Crear una reparación**

1. Usuario llena formulario en `RepairForm.vue`
2. Componente valida datos localmente
3. Llama a `useRepairs().createRepair(data)`
4. Composable hace POST a `/api/v1/repairs`
5. Endpoint `repairs.py` valida con Pydantic schema
6. Verifica autenticación y permisos
7. Llama a `repair.create()` del CRUD
8. SQLAlchemy inserta en tabla `repairs`
9. Event system dispara hook de auditoría
10. Se registra en tabla `audit_logs`
11. Email service envía notificación al cliente
12. Backend retorna objeto creado
13. Frontend actualiza store de reparaciones
14. UI refleja cambio reactivamente

11.3 Gestión de estado (Frontend)

Herramienta: Pinia

Stores implementados:

- `auth.js`: Estado de autenticación (user, token, isAuthenticated)
- `inventory.js`: Items de inventario, filtros, paginación
- `repairs.js`: Reparaciones, estados, filtros
- `categories.js`: Categorías y taxonomía
- `diagnostics.js`: Diagnósticos e imágenes
- `instruments.js`: Catálogo de instrumentos
- `quotation.js`: Cotizaciones activas
- `users.js`: Usuarios del sistema
- `stockMovements.js`: Movimientos de inventario

Patrón de uso:

```javascript
// En un componente
import { useRepairsStore } from '@/stores/repairs'

const repairsStore = useRepairsStore()
await repairsStore.fetchRepairs()
const myRepairs = repairsStore.repairs
```

11.4 Comunicación Frontend-Backend

Herramienta: Axios encapsulado en `useApi.js`

Features:

- Interceptores para agregar token JWT automáticamente
- Manejo centralizado de errores
- Loading states
- Retry logic
- Timeout configuration

11.5 Análisis de imágenes con IA

Backend: `ai_detector.py`

Posible implementación:

- Modelo de computer vision (TensorFlow/PyTorch)
- Detección de componentes electrónicos
- Clasificación de fallas visuales
- OCR para números de serie
- Generación de bounding boxes

Limitación detectada: No se encontró implementación detallada del modelo de IA en el código proporcionado.

---

12. Qué Pretende Hacer

12.1 Objetivos del proyecto

1. **Digitalizar el proceso de reparación:**
    - Eliminar papeles y hojas de cálculo
    - Centralizar información en una base de datos
    - Automatizar flujos de trabajo
2. **Mejorar experiencia del cliente:**
    - Transparencia en el proceso
    - Actualizaciones en tiempo real
    - Cotizaciones rápidas y precisas
    - Self-service para tracking
3. **Optimizar operaciones:**
    - Gestión eficiente de inventario
    - Alertas de stock bajo
    - Tracking de movimientos
    - Reportes y estadísticas
4. **Aprovechar IA para diagnósticos:**
    - Reducir tiempo de diagnóstico inicial
    - Mejorar precisión en detección de fallas
    - Generar cotizaciones más exactas
    - Documentar visualmente las reparaciones
5. **Escalabilidad:**
    - Base para futuras funcionalidades
    - Potencial multi-tenant (múltiples talleres)
    - API reutilizable

12.2 Funcionalidades proyectadas (no implementadas completamente)

Basado en la estructura:

- **Sistema de agendamiento** (`scheduler.js` en composables)
- **Notificaciones push** (estructura básica presente)
- **Reportes avanzados** (stats.py parcial)
- **Sistema de pagos online** (modelo Payment presente, pero sin integración)
- **Chat o mensajería** (no encontrado, pero sería útil)
- **App móvil** (estructura preparada para PWA)

13. En Qué Se Queda Corto

13.1 Módulo de IA - Implementación Incompleta

Problema:

- El código tiene la estructura (`ai_detector.py`, componentes de UI), pero no hay evidencia de un modelo de IA real funcionando
- No se encontraron pesos del modelo, pipelines de inferencia o integraciones con servicios de IA

Impacto:

- La funcionalidad principal diferenciadora no está operativa
- Las cotizaciones automáticas no pueden generarse con precisión

Recomendación:

- Integrar con Google Cloud Vision API, AWS Rekognition o Azure Computer Vision
- Entrenar un modelo custom con dataset de instrumentos dañados
- Implementar pipeline de inferencia con cache de resultados

13.2 Sistema de Pagos - No Integrado

Problema:

- Existe el modelo `Payment` y endpoints básicos
- No hay integración con pasarelas de pago (Stripe, PayPal, Transbank para Chile)
- No hay webhooks para confirmación de pagos

Impacto:

- Los clientes no pueden pagar online
- El flujo de pago debe hacerse fuera del sistema

Recomendación:

- Integrar Transbank WebPay (estándar en Chile)
- Implementar webhooks para confirmación automática
- Agregar conciliación bancaria

13.3 Testing - Cobertura Insuficiente

Problema:

- Solo 2 tests de frontend (InventoryCard, InventoryForm)
- Backend tiene tests básicos pero no cubre todos los endpoints
- No hay tests de integración end-to-end
- No hay tests de carga o performance

Impacto:

- Riesgo alto de regressions
- Difícil refactorizar con confianza
- No se conoce el comportamiento bajo carga

Recomendación:

- Alcanzar mínimo 80% de cobertura en backend
- Agregar tests E2E con Playwright o Cypress
- Implementar tests de carga con Locust o k6

13.4 Documentación API - Ausente

Problema:

- FastAPI genera docs automáticas (Swagger), pero no hay documentación escrita
- No hay ejemplos de uso de la API
- No hay guía de integración para desarrolladores externos

Impacto:

- Difícil para nuevos desarrolladores entender la API
- Imposible para terceros integrarse

Recomendación:

- Documentar todos los endpoints con ejemplos
- Crear Postman collection
- Agregar guía de autenticación y permisos

13.5 Internacionalización - No Implementada

Problema:

- Todo el texto está hardcodeado en español
- Existe `strings.js` pero no hay sistema i18n real
- No hay soporte para múltiples idiomas

Impacto:

- Limitado al mercado de habla hispana
- No escalable internacionalmente

Recomendación:

- Implementar vue-i18n
- Crear archivos de traducción (es, en)
- Externalizar todos los strings

13.6 Monitoreo y Observabilidad - Mínimo

Problema:

- Logging básico implementado
- No hay métricas de performance
- No hay alertas
- No hay dashboards de salud del sistema

Impacto:

- Difícil detectar problemas en producción
- No hay visibilidad de performance
- Tiempo de respuesta a incidentes lento

Recomendación:

- Implementar Sentry para error tracking
- Agregar Prometheus + Grafana para métricas
- Configurar alertas críticas
- Implementar health checks

13.7 Mobile Experience - Limitado

Problema:

- Diseño responsive presente pero no optimizado para móvil
- No hay app nativa
- No hay PWA configurada

Impacto:

- Experiencia subóptima en smartphones
- No hay notificaciones push reales
- No funciona offline

Recomendación:

- Configurar como PWA (manifest.json, service worker)
- Optimizar UI para touch
- Implementar notificaciones push web

13.8 Backup y Disaster Recovery - Sin Implementar

Problema:

- No hay estrategia de backups
- No hay plan de recuperación ante desastres
- No hay replicación de base de datos

Impacto:

- Riesgo catastrófico de pérdida de datos
- Sin plan B ante fallas de servidor

Recomendación:

- Implementar backups automáticos diarios
- Configurar replicación de PostgreSQL
- Documentar procedimientos de recuperación
- Testear restauración periódicamente

13.9 Performance - No Optimizado

Problemas detectados:

- No hay cache en consultas frecuentes
- Imágenes no optimizadas
- No hay lazy loading de componentes
- No hay paginación en algunos listados

Impacto:

- Tiempos de carga lentos
- Uso excesivo de recursos
- Mala experiencia de usuario con bases de datos grandes

Recomendación:

- Implementar Redis para cache
- Optimizar imágenes (WebP, lazy loading)
- Agregar code splitting en Vue
- Virtualizar listados largos (vue-virtual-scroller)

13.10 CI/CD - Pipeline Incompleto

Problema:

- Existe carpeta `.github/workflows/` pero está vacía o incompleta
- No hay deployment automático
- No hay ambientes de staging
- No hay rollback automático

Impacto:

- Deployments manuales propensos a errores
- No hay validación pre-producción
- Tiempo de deploy lento

Recomendación:

- Implementar pipeline completo (lint → test → build → deploy)
- Configurar ambiente de staging
- Implementar blue-green deployment
- Agregar health checks post-deploy

---

14. Deuda Técnica Identificada

14.1 Alta prioridad

1. **Implementar módulo de IA real** - Core feature no funcional
2. **Integrar sistema de pagos** - Bloqueante para monetización
3. **Aumentar cobertura de tests** - Riesgo de producción
4. **Implementar backups** - Riesgo catastrófico

14.2 Media prioridad

5. **Documentación API completa** - Bloqueante para integraciones
6. **Sistema de monitoreo** - Visibilidad de producción
7. **Optimización de performance** - UX degradada
8. **Configurar CI/CD** - Eficiencia del equipo

14.3 Baja prioridad

9. **Internacionalización** - Expansión futura
10. **PWA y mobile** - Mejora de UX
11. **Refactoring de código legacy** - Mantenibilidad

---

15. Fortalezas del Proyecto

15.1 Arquitectura Sólida

✅ Separación clara de responsabilidades (frontend/backend)
✅ Uso de mejores prácticas (Composition API, async/await)
✅ Modular y escalable
✅ Type safety con Pydantic

15.2 Stack Moderno

✅ Tecnologías actuales y bien soportadas
✅ Ecosystem maduro (Vue 3, FastAPI)
✅ Performance potencial excelente

15.3 Features Completas en Admin

✅ Panel administrativo robusto
✅ Gestión de inventario completa
✅ Sistema de categorías flexible
✅ Tracking de reparaciones detallado

15.4 Seguridad Básica Implementada

✅ JWT authentication
✅ Password hashing
✅ Rate limiting
✅ Input validation
✅ Sistema de roles

15.5 Auditoría Completa

✅ Event system para tracking
✅ Logs estructurados
✅ Historial de cambios

---

16. Roadmap Sugerido

Fase 1: Estabilización (1-2 meses)

- [ ] Implementar backups automáticos
- [ ] Aumentar cobertura de tests a 80%+
- [ ] Configurar monitoreo básico (Sentry)
- [ ] Documentar API con ejemplos

Fase 2: Features Core (2-3 meses)

- [ ] Integrar IA real o servicio externo
- [ ] Implementar sistema de pagos (Transbank)
- [ ] Optimizar performance (cache, lazy loading)
- [ ] Configurar CI/CD completo

Fase 3: Mejoras UX (1-2 meses)

- [ ] Configurar como PWA
- [ ] Optimizar mobile experience
- [ ] Implementar notificaciones push
- [ ] Agregar internacionalización

Fase 4: Escalabilidad (ongoing)

- [ ] Implementar multi-tenancy
- [ ] Agregar analytics avanzados
- [ ] API pública para integraciones
- [ ] Sistema de reportes avanzados

---

17. Estimación de Esfuerzo

Recursos necesarios para completar features críticas:

Tarea	Esfuerzo	Prioridad
Integración IA real	3-4 semanas	Alta
Sistema de pagos	2-3 semanas	Alta
Tests completos	2-3 semanas	Alta
Backups y DR	1 semana	Alta
Documentación API	1-2 semanas	Media
Monitoreo	1 semana	Media
Optimización performance	2-3 semanas	Media
CI/CD	1 semana	Media
PWA + mobile	2-3 semanas	Baja
Internacionalización	1-2 semanas	Baja

Total estimado: 16-24 semanas de desarrollo

---

18. Conclusiones

El proyecto tiene:

✅ **Bases sólidas:**

- Arquitectura bien estructurada
- Stack tecnológico adecuado
- Features administrativas completas
- Código mayormente limpio y organizado

⚠️ **Gaps críticos:**

- IA no implementada (feature principal)
- Sistema de pagos sin integrar
- Testing insuficiente
- Sin estrategia de backups

🔄 **Necesita:**

- 4-6 meses de desarrollo adicional
- 1-2 desarrolladores full-time
- Presupuesto para servicios externos (IA, pagos, hosting)

Viabilidad del proyecto:

Para MVP básico: ✅ **Listo** (landing + admin funcional)
Para producción completa: ⚠️ **Requiere 4-6 meses más**
Para escalabilidad: ❌ **Necesita refactoring significativo**

---

19. Recomendaciones Finales

Corto plazo (1 mes):

1. Implementar backups INMEDIATAMENTE
2. Aumentar tests a nivel crítico
3. Configurar monitoreo básico
4. Documentar endpoints principales

Mediano plazo (3-6 meses):

5. Integrar IA o servicio externo equivalente
6. Implementar sistema de pagos
7. Optimizar performance
8. Completar CI/CD

Largo plazo (6-12 meses):

9. PWA y mobile optimization
10. Internacionalización
11. Multi-tenancy
12. API pública

20. Contacto y Recursos

Repositorio: https://github.com/CristobalZurita/cirujano-front
Website: www.cirujanodesintetizadores.cl
Stack: Vue 3 + Vite + FastAPI + PostgreSQL
Licencia: [Verificar en repositorio]

Documento generado el 8 de enero de 2026
Versión: 1.0
Autor del análisis: Claude (Anthropic)