SISTEMA DE SALIDA DE VEHÍCULOS CHILE-ARGENTINA

**(DAS) Documento Arquitectura de Software Versión 1.0**



**Identificación de Documento**

| **Identificación** | **DAS-SVA-001** |
| --- | --- |
| **Proyecto** | Sistema de Salida de Vehículos Chile-Argentina |
| **Versión** | 6.0 |

| **Documento mantenido por** | **Equipo de Desarrollo** |
| --- | --- |
| **Fecha de última revisión** | 05/07/2025 |
| **Fecha de próxima revisión** | 07/07/2025 |

| **Documento aprobado por** | **Ricardo Alfonso** |
| --- | --- |
| **Fecha de última aprobación** | 05/07/2025 |



**Historia de Revisiones**

| **Fecha** | **Versión** | **Descripción** | **Autor** |
| --- | --- | --- | --- |
| 26/06/2025  05/07/2025 | 1.0  6.0 | Creación inicial del documento  Versión Final del Documento | Nicolás Fernández, Diego Iturra, Ignacio Valeria  Nicolás Fernández, Diego Iturra, Ignacio  Valeria |



**Integrantes del Proyecto**

* **Nicolás Fernández Vera**
* **Diego Iturra**
* **Ignacio Valeria Docente:** Ricardo Alfonso

**Sección:** 002-D

**Fecha:** 26/05/2025



# INTRODUCCIÓN

## Contexto del Problema

El control de salida de vehículos entre Chile y Argentina representa un desafío crítico para las autoridades fronterizas, debido a la necesidad de gestionar eficientemente el flujo vehicular, verificar documentación, controlar el cumplimiento de normativas aduaneras y mantener registros precisos de los movimientos transfronterizos.

Actualmente, los procesos manuales o sistemas desactualizados generan demoras, errores en la documentación y dificultades para el seguimiento en tiempo real de los vehículos que cruzan la frontera. Esto impacta directamente en la experiencia del usuario y en la eficiencia operativa de los puestos de control fronterizo.

## Propósito

Este documento tiene como propósito definir la arquitectura de software del Sistema de Salida de Vehículos Chile-Argentina, estableciendo las bases técnicas y funcionales para su desarrollo e implementación. Describe los componentes del sistema, sus interacciones, patrones arquitectónicos y requisitos de calidad.

El documento sirve como guía para desarrolladores, arquitectos de software, stakeholders y personal técnico involucrado en el proyecto, proporcionando una visión integral del sistema desde múltiples perspectivas arquitectónicas.

## Ámbito

El sistema abarca la gestión completa del proceso de salida de vehículos desde Chile hacia Argentina, incluyendo:

* Registro y verificación de vehículos
* Validación de documentación (pasaportes, permisos, documentación vehicular)
* Control aduanero y verificación de mercancías
* Generación de permisos de salida
* Integración con sistemas gubernamentales de ambos países
* Registro histórico y reportería
* Notificaciones y alertas automatizadas

El sistema no incluye procesos de salida de vehículos ni otros tipos de control fronterizo no relacionados con vehículos particulares o comerciales.

## Definiciones, acrónimos y abreviaciones

| **ACRÓNIMO** | **DESCRIPCIÓN** |
| --- | --- |
| **DAS** | Documento de Arquitectura de Software |
| **SVA** | Sistema de Salida de Vehículos Argentina |
| **API** | Application Programming Interface |
| **CRUD** | Create, Read, Update, Delete |
| **UI** | User Interface |
| **UX** | User Experience |
| **PDV** | Punto de Venta/Control |
| **PDA** | Personal Digital Assistant |
| **WCAG** | Web Content Accessibility Guidelines |

* 1. **Resumen ejecutivo**

El Sistema de Salida de Vehículos Chile-Argentina es una solución integral diseñada para modernizar y automatizar los procesos de control fronterizo vehicular. Implementa una arquitectura de microservicios que permite escalabilidad, mantenibilidad y alta disponibilidad.

El sistema integra múltiples componentes: interfaz web responsiva, aplicaciones móviles para inspectores, APIs para integración con sistemas gubernamentales, base de datos distribuida y módulos de inteligencia artificial para detección automática de anomalías.

Los beneficios esperados incluyen reducción del tiempo de procesamiento en un 60%, eliminación de errores manuales, mejora en la trazabilidad de vehículos y cumplimiento automático de normativas internacionales.

## Arquitectura del sistema

La arquitectura del sistema se basa en el modelo 4+1 de Kruchten, contemplando las siguientes vistas:

* **Vista de Escenario**: Casos de uso principales del sistema
* **Vista Lógica**: Organización conceptual del software en paquetes y clases
* **Vista de Desarrollo**: Organización del software en módulos y componentes
* **Vista de Proceso**: Aspectos dinámicos del sistema en tiempo de ejecución
* **Vista Física**: Distribución del software en los elementos de hardware



# VISIÓN DEL SISTEMA

## Descripción general del sistema

El Sistema de Salida de Vehículos Chile-Argentina es una plataforma digital integral que automatiza y gestiona todos los procesos relacionados con la salida de vehículos desde territorio chileno hacia Argentina.

El sistema opera en tiempo real, proporcionando interfaces intuitivas para diferentes tipos de usuarios: conductores, inspectores fronterizos, funcionarios aduaneros y administradores del sistema. Integra tecnologías de reconocimiento automático de placas, validación biométrica, verificación documental digital y sistemas de alerta inteligente.

La plataforma está diseñada para operar 24/7 con alta disponibilidad, soportando picos de tráfico durante temporadas turísticas y manteniendo la seguridad de la información mediante encriptación de extremo a extremo.

## Objetivos del sistema Objetivos Primarios:

* Reducir el tiempo promedio de procesamiento de salida vehicular de 15 a 6 minutos
* Eliminar errores manuales en el procesamiento de documentación
* Mejorar la trazabilidad y seguimiento de vehículos en tránsito
* Facilitar la integración con sistemas argentinos para procesos continuos

## Objetivos Secundarios:

* Generar reportes automáticos para análisis de flujo vehicular
* Implementar alertas proactivas para situaciones irregulares
* Mejorar la experiencia del usuario en el proceso de cruce fronterizo
* Cumplir con normativas de seguridad y privacidad de datos de ambos países

## Objetivos de Calidad:

* Disponibilidad del sistema superior al 99.5%
* Tiempo de respuesta menor a 3 segundos para operaciones críticas
* Capacidad de procesamiento de hasta 1000 vehículos simultáneos
* Cumplimiento total con estándares WCAG 2.1 nivel AA

## Principales funcionalidades esperadas Gestión de Vehículos:

* Registro automático mediante reconocimiento de placas
* Verificación de documentación vehicular (revisión técnica, seguro, permisos)
* Validación de antecedentes y restricciones de circulación
* Generación de códigos QR para seguimiento

## Gestión de Conductores:

* Verificación de identidad mediante documento y/o biometría
* Validación de licencia de conducir y vigencia
* Control de antecedentes penales y restricciones de salida
* Registro de acompañantes cuando corresponda

## Control Aduanero:

* Declaración de mercancías transportadas
* Verificación de restricciones de exportación
* Cálculo automático de aranceles cuando aplique
* Generación de documentos aduaneros digitales

## Integración y Comunicación:

* Sincronización con sistemas argentinos de ingreso
* Notificaciones automáticas a autoridades competentes
* Alertas en tiempo real para situaciones de riesgo
* Integración con sistemas de inteligencia y seguridad

## Reportería y Analytics:

* Dashboards en tiempo real de flujo vehicular
* Reportes estadísticos por períodos configurables
* Análisis predictivo de patrones de tráfico
* Informes de cumplimiento normativo

## Supuestos y dependencias Supuestos:

* Disponibilidad de conectividad a internet estable en puntos fronterizos
* Cooperación técnica entre autoridades chilenas y argentinas
* Capacitación adecuada del personal operativo
* Infraestructura básica de hardware disponible en puestos de control

## Dependencias Internas:

* Desarrollo e implementación de APIs de integración
* Configuración de base de datos distribuida
* Implementación de sistemas de seguridad y encriptación
* Desarrollo de interfaces de usuario responsivas

## Dependencias Externas:

* APIs de sistemas gubernamentales chilenos (Registro Civil, Carabineros, SII)
* APIs de sistemas gubernamentales argentinos (Migraciones, AFIP, Gendarmería)
* Servicios de terceros para validación documental y biométrica
* Proveedores de infraestructura cloud y servicios de telecomunicaciones

## Riesgos Identificados:

* Cambios en normativas fronterizas que requieran modificaciones al sistema
* Indisponibilidad temporal de sistemas gubernamentales externos
* Picos de tráfico excepcionales que superen la capacidad planificada
* Requerimientos de seguridad adicionales posterior a la implementación



# ESTILOS Y PATRONES ARQUITECTÓNICOS

## Estilo arquitectónico adoptado

El sistema implementa una **arquitectura de microservicios** como estilo principal, complementada con elementos de **arquitectura orientada a eventos** para el manejo de notificaciones y sincronización en tiempo real.

Los microservicios están organizados por dominio funcional:

* Servicio de Gestión de Vehículos
* Servicio de Gestión de Usuarios
* Servicio de Control Documental
* Servicio de Integración Aduanera
* Servicio de Notificaciones
* Servicio de Reportería y Analytics
* Servicio de Seguridad y Autenticación

## Justificación del estilo según el contexto del sistema Ventajas de Microservicios para este contexto:

1. **Escalabilidad independiente**: Permite escalar componentes específicos según demanda (ej: más instancias del servicio de verificación documental durante horarios pico)
2. **Resiliencia**: El fallo de un microservicio no compromete la operación completa del sistema
3. **Integración facilitada**: Cada microservicio puede integrarse independientemente con sistemas externos específicos
4. **Desarrollo paralelo**: Diferentes equipos pueden trabajar simultáneamente en distintos servicios
5. **Tecnología heterogénea**: Permite usar diferentes tecnologías según las necesidades específicas de cada servicio
6. **Mantenimiento simplificado**: Actualizaciones y correcciones se pueden aplicar a servicios individuales sin afectar el conjunto

**Arquitectura orientada a eventos** complementa los microservicios proporcionando:

* + Comunicación asíncrona entre servicios
  + Desacoplamiento temporal entre componentes
  + Capacidad de reacción inmediata a eventos críticos
  + Facilidad para implementar auditoría completa del sistema

## Patrones de diseño aplicados Patrón API Gateway:

* Punto único de salida para todas las operaciones externas
* Manejo centralizado de autenticación, autorización y rate limiting
* Enrutamiento inteligente hacia microservicios apropiados

## Patrón CQRS (Command Query Responsibility Segregation):

* Separación entre operaciones de escritura y lectura
* Optimización independiente para consultas y comandos
* Simplificación de modelos de datos complejos

## Patrón Repository:

* Abstracción de la capa de acceso a datos
* Facilita testing mediante mock repositories
* Permite cambio de tecnología de persistencia sin afectar lógica de negocio

## Patrón Observer:

* Implementación de notificaciones automáticas
* Desacoplamiento entre emisores y receptores de eventos
* Facilita extensibilidad para nuevos tipos de alertas

## Patrón Circuit Breaker:

* Protección contra fallos en servicios externos
* Implementación de timeouts y reintentos automáticos
* Degradación elegante de funcionalidades



# MODELO 4+1 Y VISTAS ARQUITECTÓNICAS

* 1. **VISTA DE ESCENARIO**

## Propósito

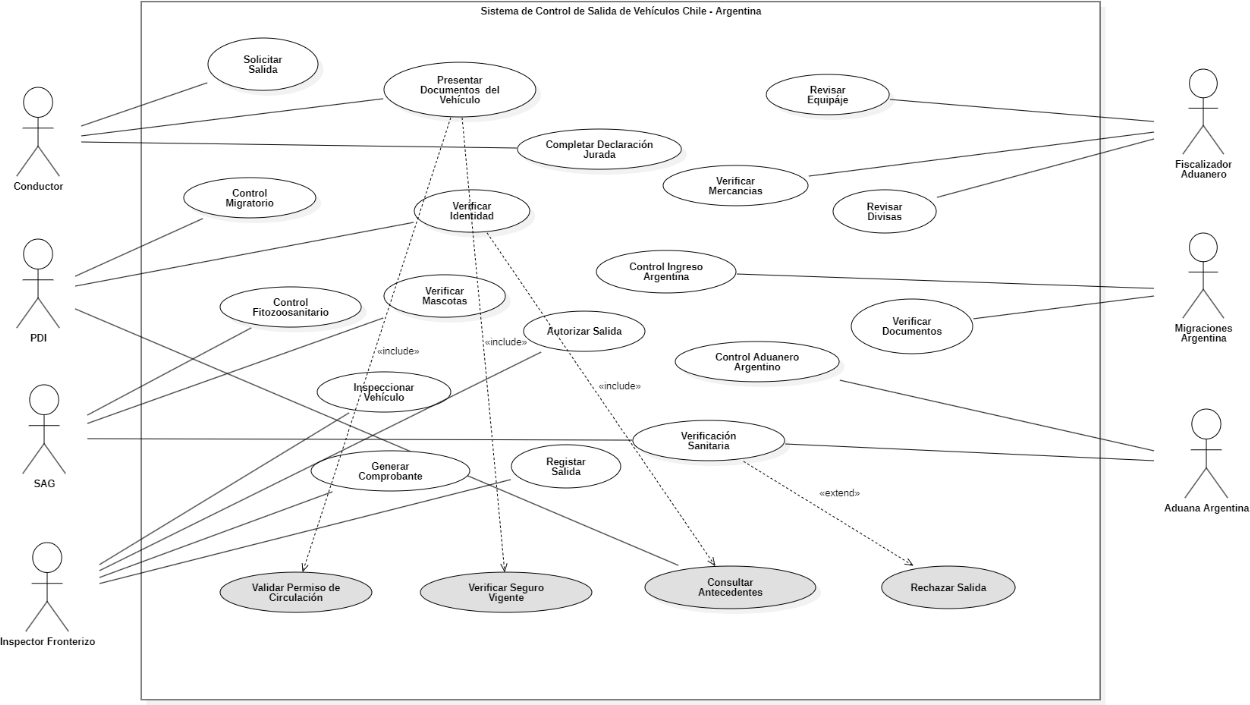
La vista de escenario describe las funcionalidades del sistema desde la perspectiva de los usuarios finales, identificando los casos de uso principales que impulsan las decisiones arquitectónicas. Esta vista sirve como punto de partida para comprender los requisitos funcionales y no funcionales del sistema.

## Actores

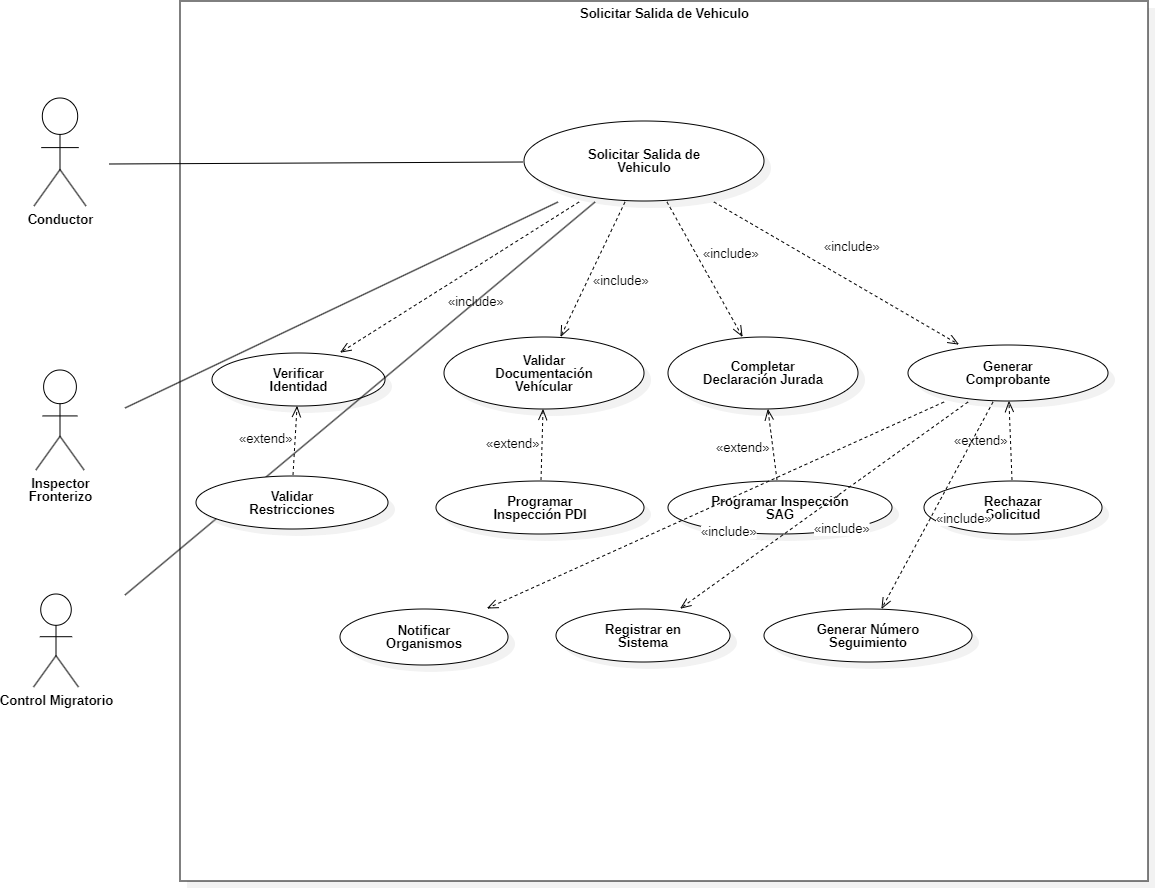
**Conductor:** Usuario que solicita salida del país con su vehículo

**Inspector Fronterizo:** Funcionario que verifica documentación y autoriza salidas **Funcionario Aduanero:** Personal encargado del control de mercancías **Administrador del Sistema:** Responsable de configuración y mantenimiento **Sistema Argentino:** Sistemas externos de Argentina que reciben información **Sistema Gubernamental Chileno:** APIs de organismos públicos chilenos

## Diagrama general de casos de uso

Este diagrama fue incluido porque corresponde a la Vista de Escenarios del modelo 4+1. Representa los principales comportamientos esperados del sistema según los actores involucrados (Inspector, Conductor, Sistema Chileno/Argentino). Su propósito es mostrar los requerimientos funcionales clave de forma clara y accesible para todos los stakeholders.

## Diagrama de casos de uso específicos

Este diagrama fue elegido porque representa visualmente [comportamiento/estructura/proceso] del sistema. Permite al equipo de desarrollo comprender [relaciones entre clases / flujo de interacción / arquitectura física], facilitando la implementación modular y mantenible del sistema.

* + 1. **Lista de casos de uso**

| **Código** | **Nombre** | **Actores** |
| --- | --- | --- |
| **CU-001-**  **001** | Registrar solicitud de salida  vehicular | Conductor, Inspector Fronterizo |
| **CU-001-**  **002** | Verificar documentación personal | Inspector Fronterizo, Sistema  Gubernamental |
| **CU-001-**  **003** | Verificar documentación vehicular | Inspector Fronterizo, Sistema  Gubernamental |
| **CU-001- 004** | Realizar control aduanero | Funcionario Aduanero, Sistema Aduanero |
| **CU-001-**  **005** | Generar autorización de salida | Inspector Fronterizo |
| **CU-001-**  **006** | Sincronizar con sistema argentino | Sistema Argentino |
| **CU-002-**  **001** | Consultar historial de salidas | Administrador del Sistema |
| **CU-002- 002** | Generar reportes estadísticos | Administrador del Sistema |
| **CU-002-**  **003** | Configurar alertas del sistema | Administrador del Sistema |

| **CU-003-**  **001** | Gestionar usuarios del sistema | Administrador del Sistema |
| --- | --- | --- |

* + 1. **Especificación de casos de uso**

**CU-001-001: Registrar solicitud de salida vehicular**

| **Caso de Uso** | **Registrar solicitud de salida vehicular** | **Identificador** |
| --- | --- | --- |
| **Actores** | Conductor, Inspector Fronterizo | |
| **Tipo** | Primario | |
| **Referencias** | RF-001, RF-002, RF-003 | |
| **Precondición** | El vehículo debe estar presente en el puesto fronterizo y el  conductor debe portar documentación requerida | |
| **Postcondición** | Se registra la solicitud en el sistema y se inicia el proceso de  verificación | |
| **Descripción** | El conductor se presenta en el puesto fronterizo solicitando  autorización para salir del país con su vehículo | |
| **Resumen** | Proceso inicial donde se capturan datos básicos del vehículo  y conductor para iniciar la verificación de salida | |

# CURSO NORMAL

| **Nro.** | **Ejecutor** | **Paso o Actividad** |
| --- | --- | --- |
| **1** | Conductor | Se presenta en el puesto fronterizo con vehículo y documentación |
| **2** | Inspector  Fronterizo | Accede al sistema de salida de vehículos |
| **3** | Sistema | Presenta formulario de registro de nueva solicitud |
| **4** | Inspector  Fronterizo | Escanea código de placa vehicular o ingresa manualmente |
| **5** | Sistema | Reconoce automáticamente datos del vehículo mediante  cámaras |
| **6** | Inspector  Fronterizo | Solicita documentos de identidad al conductor |
| **7** | Conductor | Entrega cédula de identidad o pasaporte |
| **8** | Inspector  Fronterizo | Escanea documento de identidad |
| **9** | Sistema | Registra datos básicos y asigna número de solicitud |
| **10** | Sistema | Inicia procesos paralelos de verificación documental |
| **11** | Sistema | Emite comprobante temporal al conductor |
| **CURSO** | **ALTERNATIVO** | |
| **Nro.** | **Descripción de acciones alternas** | |
| **4** | Si el reconocimiento automático falla, el inspector ingresa manualmente los datos de  la placa | |
| **5** | Si las cámaras no funcionan, el inspector registra manualmente los datos del vehículo | |

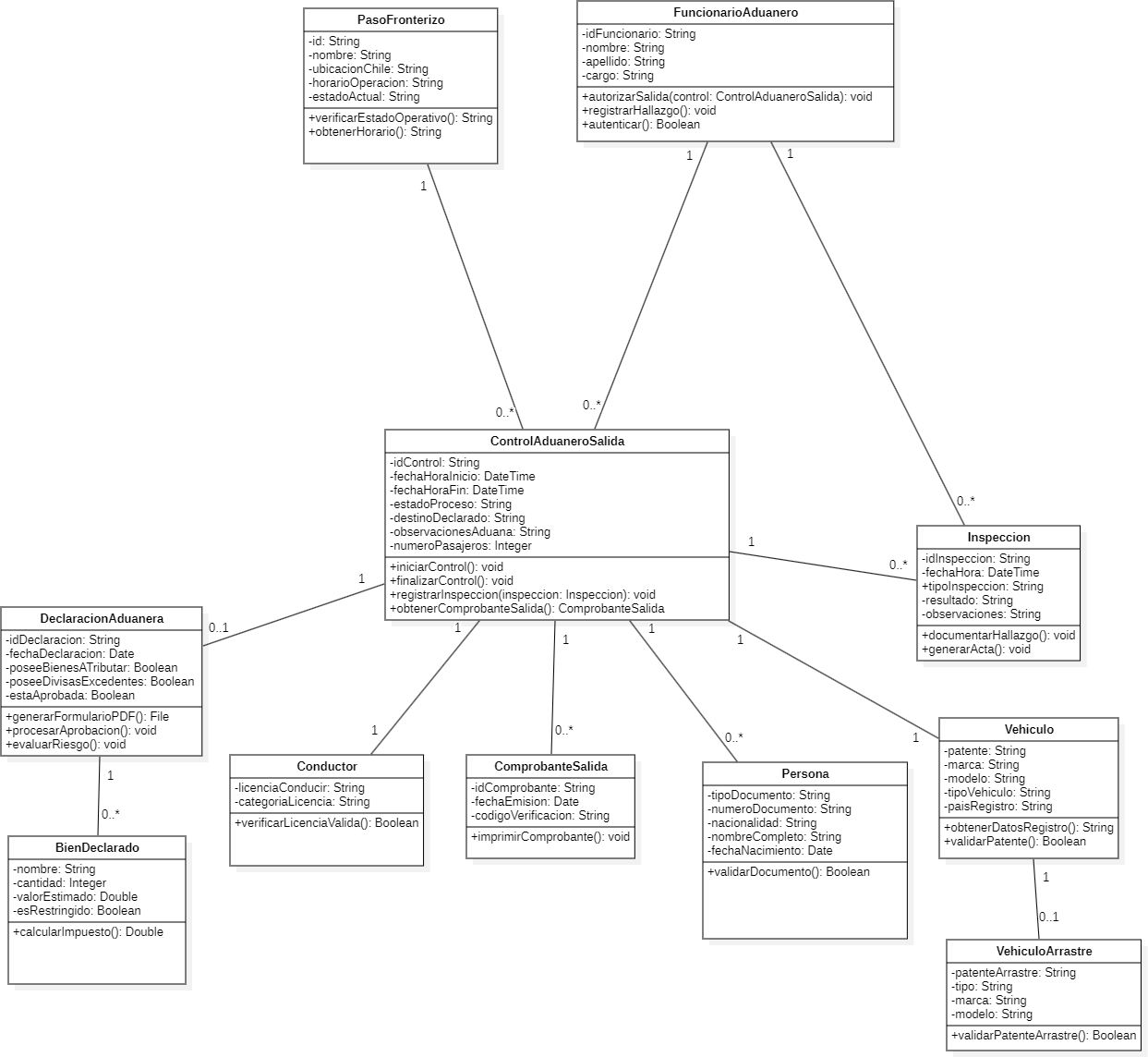
| **8** | Si el escáner no funciona, el inspector ingresa manualmente los datos del documento |
| --- | --- |
| **9** | Si hay problemas de conectividad, el sistema opera en modo offline y sincroniza posteriormente |

* 1. **VISTA LÓGICA**

## Propósito

La vista lógica describe la organización conceptual del software en términos de capas, paquetes, clases y sus relaciones. Esta vista se centra en los servicios que el sistema debe proveer a los usuarios finales y cómo estos servicios están estructurados lógicamente.

## Diagrama de clases

Este diagrama corresponde a la Vista Lógica. Se utiliza para representar la estructura estática del sistema, detallando las clases, sus atributos, métodos y relaciones. Su inclusión permite comprender la organización del código fuente, facilitando la implementación coherente de los requisitos del dominio.

## Descripción diagrama de clases Paquete de Entidades de Dominio:

* + - * **Vehiculo**: Representa un vehículo con atributos como placa, marca, modelo, año, tipo
      * **Conductor**: Información personal del conductor incluyendo identificación y licencia
      * **SolicitudSalida**: Entidad principal que agrupa toda la información de una solicitud
      * **DocumentoVehicular**: Abstracción para diferentes tipos de documentos del vehículo
      * **ControlAduanero**: Información relacionada con el control de mercancías

## Paquete de Servicios de Negocio:

* + - * **ServicioVerificacion**: Lógica para validar documentación y antecedentes
      * **ServicioIntegracion**: Manejo de comunicación con sistemas externos
      * **ServicioNotificacion**: Gestión de alertas y comunicaciones automáticas
      * **ServicioReporte**: Generación de informes y estadísticas

## Paquete de Acceso a Datos:

* + - * **RepositorioVehiculo**: Persistencia y consulta de datos vehiculares
      * **RepositorioSolicitud**: Gestión de solicitudes de salida
      * **RepositorioUsuario**: Manejo de usuarios del sistema

## Paquete de Interfaces Externas:

* + - * **InterfazSistemaChileno**: Comunicación con APIs gubernamentales chilenas
      * **InterfazSistemaArgentino**: Integración con sistemas argentinos
      * **InterfazNotificacion**: Servicios de mensajería y alertas



# VISTA DE IMPLEMENTACIÓN/DESARROLLO

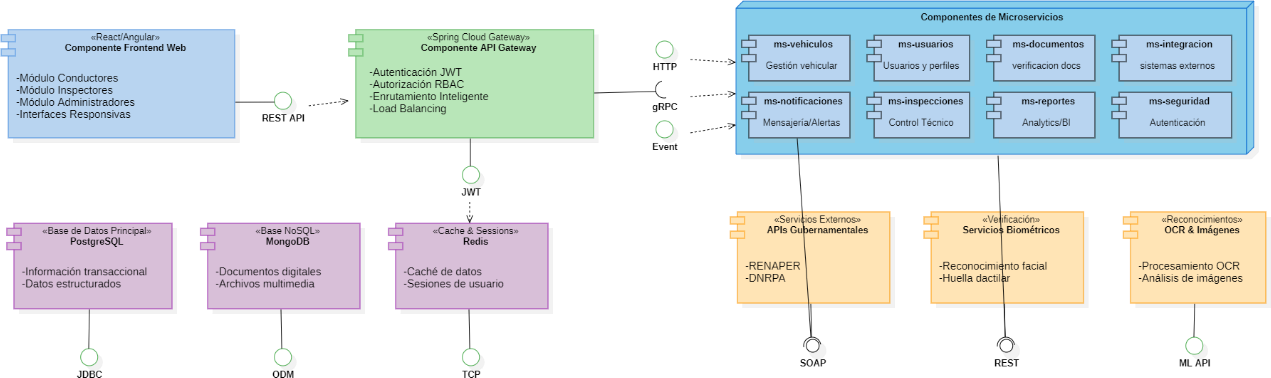
## Propósito

Esta vista describe la organización del software desde la perspectiva del desarrollador, mostrando la descomposición del sistema en componentes de implementación y sus dependencias. Define la estructura de módulos, librerías y subsistemas que conforman la arquitectura física del código.

## Diagrama de componente

Este diagrama fue elegido porque representa visualmente [comportamiento/estructura/proceso] del sistema. Permite al equipo de desarrollo comprender [relaciones entre clases / flujo de interacción / arquitectura física], facilitando la

implementación modular y mantenible del sistema.



## Descripción diagrama de componente Componente Frontend Web:

* + - * Implementado en React/Angular para interfaces responsivas
      * Incluye módulos para conductores, inspectores y administradores
      * Comunicación mediante APIs REST con el backend

## Componente API Gateway:

* + - * Implementado con Spring Cloud Gateway o similar
      * Maneja autenticación JWT y autorización RBAC
      * Enrutamiento inteligente hacia microservicios

## Componentes de Microservicios:

* + - * **ms-vehiculos**: Gestión de información vehicular
      * **ms-usuarios**: Manejo de usuarios y perfiles
      * **ms-documentos**: Verificación y almacenamiento documental
      * **ms-integracion**: Comunicación con sistemas externos
      * **ms-notificaciones**: Servicios de mensajería y alertas

## Componente de Base de Datos:

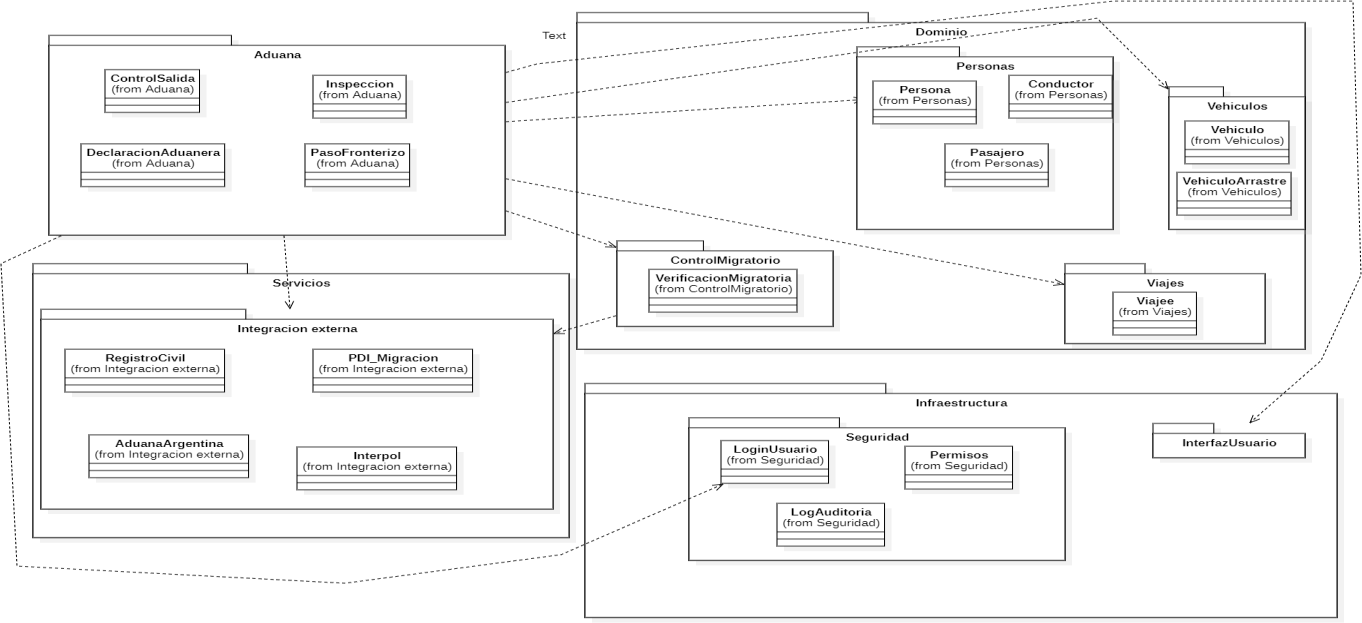
* + - * Base datos PostgreSQL para información transaccional
      * MongoDB para documentos y archivos digitales
      * Redis para caché y sesiones de usuario

## Componente de Servicios Externos:

* + - * Interfaces hacia APIs gubernamentales
      * Servicios de verificación biométrica
      * Sistemas de reconocimiento de imágenes

## Diagrama de paquete

Este diagrama se enmarca en la Vista de Desarrollo y representa la organización lógica del código fuente en paquetes. Su inclusión facilita la separación de responsabilidades, la reutilización de código y el mantenimiento a largo plazo, mostrando cómo se estructura internamente el proyecto según capas y dominios funcionales.



## Descripción diagrama de paquete Paquete Core:

* + - * **core.domain**: Entidades de dominio y reglas de negocio
      * **core.application**: Casos de uso y servicios de aplicación
      * **core.infrastructure**: Implementaciones concretas de interfaces

## Paquete Web:

* + - * **web.controllers**: Controladores REST para APIs
      * **web.security**: Configuración de seguridad y autenticación
      * **web.dto**: Objetos de transferencia de datos

## Paquete Integration:

* + - * **integration.chilean**: Adaptadores para sistemas chilenos
      * **integration.argentinian**: Adaptadores para sistemas argentinos
      * **integration.external**: Servicios de terceros

## Paquete Common:

* + - * **common.utils**: Utilidades compartidas
      * **common.exceptions**: Manejo de excepciones
      * **common.constants**: Constantes del sistema



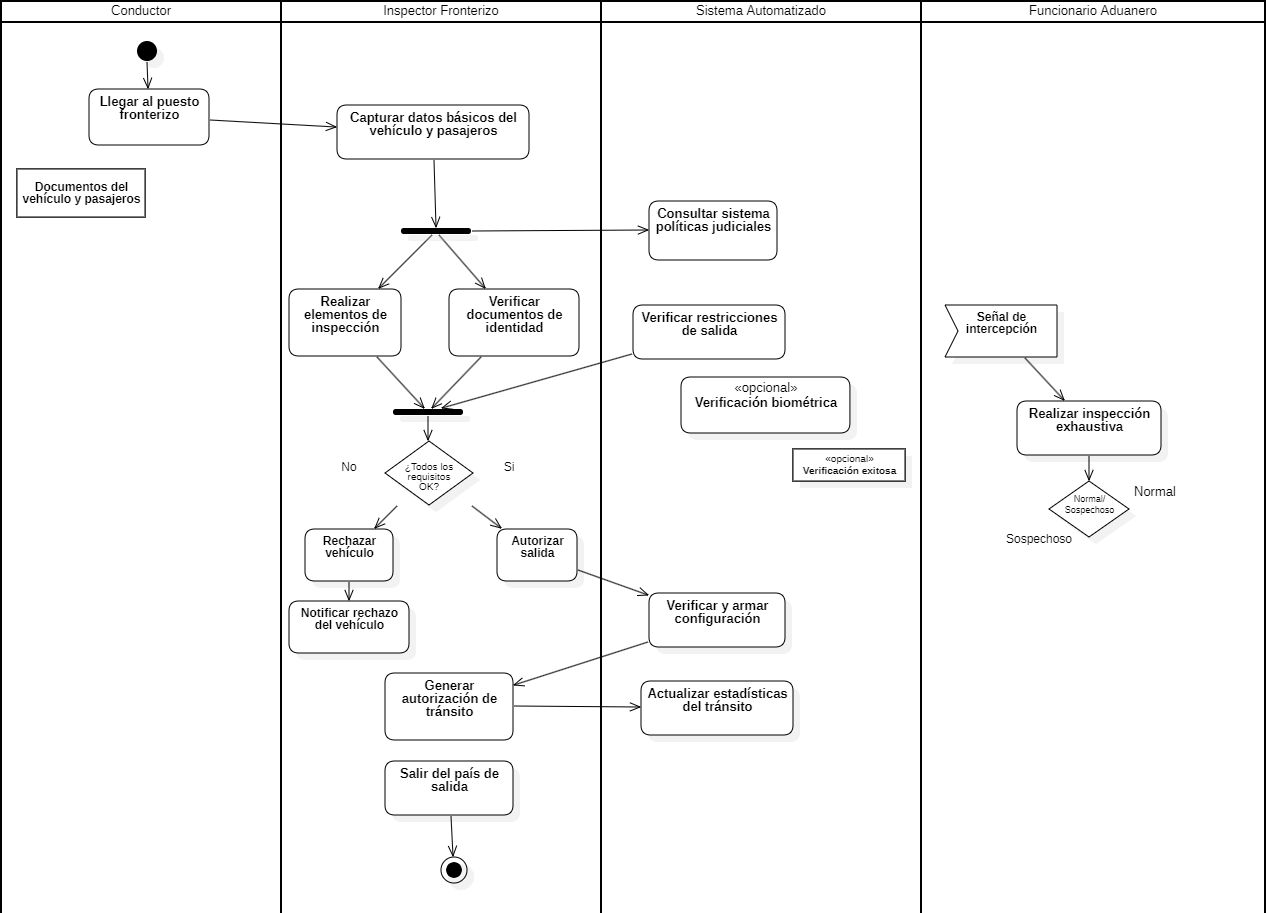
# VISTA DE PROCESOS

## Propósito

La vista de procesos describe los aspectos dinámicos del sistema, mostrando la estructura de procesos y hilos en tiempo de ejecución. Se enfoca en los aspectos de comportamiento del sistema como concurrencia, distribución, tolerancia a fallos y escalabilidad.

## Diagrama de actividad

Este diagrama se utiliza como parte de la Vista de Procesos. Representa los flujos de actividades en el sistema, incluyendo decisiones, secuencias y eventos. Su inclusión permite modelar la lógica de negocio dinámica y los distintos caminos de ejecución frente a distintas condiciones de salida.



## Descripción diagrama de actividad Proceso Principal - Salida de Vehículo:

1. **Inicio**: Conductor llega al puesto fronterizo
2. **Registro Inicial**: Sistema captura datos básicos del vehículo y conductor

## Procesos Paralelos:

* Verificación de documentación personal (subprocess)
* Verificación de documentación vehicular (subprocess)
* Control de antecedentes (subprocess)

1. **Sincronización**: Espera completar todas las verificaciones
2. **Evaluación**: Sistema determina si cumple requisitos para salida

## Decisión:

* Si aprobado: Generar autorización y notificar a Argentina
* Si rechazado: Registrar observaciones y notificar al conductor

1. **Finalización**: Registro de la transacción en historial

## Subprocesos Críticos:

* **Verificación Biométrica**: Proceso asíncrono con timeout de 30 segundos
* **Consulta a Sistemas Externos**: Implementa circuit breaker para fallos
* **Notificación Argentina**: reintentos automáticos

## Manejo de Excepciones:

* Timeouts en servicios externos activan modo degradado
* Fallos de conectividad permiten operación offline temporal
* Errores críticos generan alertas inmediatas a administradores



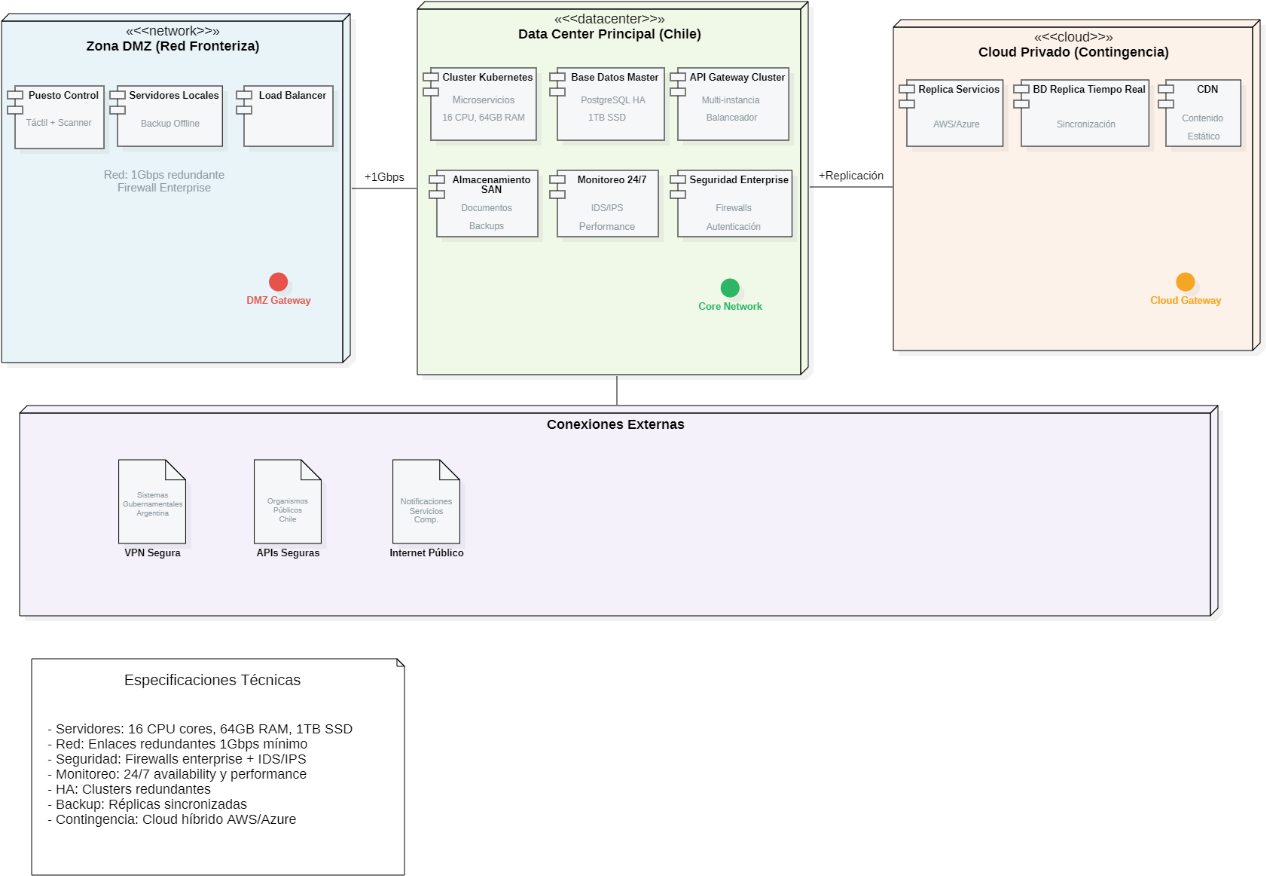
# VISTA FÍSICA

## Propósito

La vista física ilustra la distribución del software en los elementos de hardware de la infraestructura, mostrando cómo los componentes de software se mapean a nodos de procesamiento. Define la topología del sistema en términos de comunicación entre nodos.

## Diagrama de despliegue

Este diagrama forma parte de la Vista Física. Muestra la infraestructura tecnológica del sistema, especificando los nodos (servidores, contenedores) y la distribución de los componentes en estos. Su objetivo es ayudar a dimensionar el entorno de ejecución y garantizar el cumplimiento de atributos de calidad como rendimiento y disponibilidad.



## Descripción diagrama de despliegue Zona DMZ (Red Fronteriza):

* + - * **Puesto Control**: Equipos cliente con interfaces tactiles y escáneres
      * **Servidores Locales**: Backup local para operación offline
      * **Load Balancer**: Distribución de carga entre servidores

## Data Center Principal (Chile):

* + - * **Cluster Kubernetes**: Orquestación de microservicios
      * **Base Datos Master**: PostgreSQL en alta disponibilidad
      * **API Gateway Cluster**: Múltiples instancias con balanceador
      * **Almacenamiento**: SAN para documentos y backups

## Cloud Privado (Contingencia):

* + - * **Replica de Servicios**: Instancias de respaldo en AWS/Azure
      * **Base Datos Replica**: Sincronización en tiempo real
      * **CDN**: Distribución de contenido estático

## Conexiones Externas:

* + - * **VPN Segura**: Hacia sistemas gubernamentales argentinos
      * **API Seguras**: Integración con organismos públicos chilenos
      * **Internet Público**: Para notificaciones y servicios complementarios

## Especificaciones Técnicas:

* + - * Servidores: 16 CPU cores, 64GB RAM, 1TB SSD
      * Red: Enlaces redundantes de 1Gbps mínimo
      * Seguridad: Firewalls enterprise y sistemas IDS/IPS
      * Monitoreo: Herramientas 24/7 para availability y performance



# REQUISITOS DE CALIDAD

## Propósito

Definir los atributos de calidad que debe cumplir el sistema para garantizar su operación efectiva en el entorno fronterizo, estableciendo métricas específicas y métodos de evaluación para cada atributo.

* 1. **Atributos de calidad**

| **ATRIBUTO DE**  **CALIDAD** | **DESCRIPCIÓN** | **JUSTIFICACIÓN** |
| --- | --- | --- |
| **Disponibilidad** | El sistema debe estar operativo 99.5% del tiempo (máximo 36 horas de downtime anual) | Los cruces fronterizos operan 24/7 y las interrupciones afectan directamente el tráfico  internacional |
| **Rendimiento** | Tiempo de respuesta menor a 3 segundos para operaciones críticas y capacidad para 1000 usuarios  simultáneos | Los tiempos de espera prolongados generan congestión vehicular y afectan la experiencia  del usuario |
| **Seguridad** | Encriptación end-to-end, autenticación multifactor y auditoría completa de todas las operaciones | Manejo de información sensible personal y gubernamental requiere máximos estándares de  seguridad |
| **Usabilidad** | Interfaces intuitivas con tiempo de aprendizaje menor a 2 horas para  usuarios novatos | Personal rotativo y conductores ocasionales deben poder usar el  sistema sin capacitación extensa |
| **Escalabilidad** | Capacidad de incrementar recursos automáticamente hasta 300% de la  carga base | Temporadas turísticas y eventos especiales pueden triplicar el  tráfico normal |
| **Mantenibilidad** | Posibilidad de actualizar componentes sin afectar la  operación general del sistema | Actualizaciones normativas frecuentes requieren  modificaciones ágiles al sistema |
| **Portabilidad** | Funcionamiento en diferentes navegadores y dispositivos  móviles/tablets | Diversidad de equipos en puestos fronterizos y necesidad  de acceso remoto |
| **Accesibilidad** | Cumplimiento WCAG 2.1 nivel AA para usuarios con discapacidades | Garantizar acceso equitativo según normativas de inclusión  gubernamental |

* 1. **Reglas y criterios de evaluación de calidad**

**Disponibilidad:**

* Métrica: Uptime del sistema medido cada 5 minutos
* Herramientas: Prometheus + Grafana para monitoreo continuo
* Criterio: SLA de 99.5% con penalizaciones contractuales por incumplimiento

## Rendimiento:

* Métrica: Tiempo de respuesta percentil 95 < 3 segundos
* Herramientas: JMeter para pruebas de carga, New Relic para monitoreo APM
* Criterio: Pruebas semanales con 1000 usuarios virtuales simultáneos

## Seguridad:

* Métrica: Zero vulnerabilidades críticas en auditorías trimestrales
* Herramientas: SonarQube, OWASP ZAP, auditorías externas
* Criterio: Penetration testing semestral por empresa certificada

## Usabilidad:

* Métrica: Tiempo de completar tarea estándar < 5 minutos por usuario entrenado
* Herramientas: Tests de usabilidad con usuarios reales, analytics de uso
* Criterio: Satisfacción del usuario superior a 8/10 en encuestas trimestrales

## Escalabilidad:

* Métrica: Auto-scaling activado cuando CPU > 70% por 5 minutos
* Herramientas: Kubernetes HPA, métricas de infraestructura
* Criterio: Soporte verificado para 3000 usuarios simultáneos en tests anuales

## Mantenibilidad:

* Métrica: Tiempo de deployment < 15 minutos sin downtime
* Herramientas: CI/CD pipeline con Jenkins/GitLab, blue-green deployment
* Criterio: 90% de hotfixes implementados en menos de 24 horas

# PRINCIPIOS DE DISEÑO APLICADOS

## Propósito

La aplicación de principios de diseño en el desarrollo del sistema tiene como propósito garantizar la creación de un software robusto, mantenible y escalable. Estos principios actúan como guías fundamentales que determinan la estructura y comportamiento del sistema, facilitando su desarrollo, implementación y evolución a lo largo del tiempo. La correcta

aplicación de estos principios ayuda a reducir la complejidad inherente del software, mejora la calidad del código y permite adaptarse a cambios futuros con mayor facilidad.

## Principios de diseño

| **PRINCIPIO** | **DESCRIPCIÓN** | **APLICACIÓN EN EL SISTEMA** |
| --- | --- | --- |
| **Cohesión** | Cada módulo o clase tiene una única responsabilidad bien definida. | Los servicios están diseñados para realizar tareas específicas y no múltiples funciones. Por ejemplo, el servicio de gestión de vehículos se enfoca exclusivamente en las operaciones relacionadas con  vehículos. |
| **Acoplamiento bajo** | Los módulos deben tener mínimas dependencias entre sí, permitiendo que los cambios en un módulo no afecten a otros. | La arquitectura implementa interfaces bien definidas entre componentes, permitiendo que los módulos de salida de vehículos operen de manera  independiente pero coordinada. |
| **Encapsulamiento** | Ocultar los detalles internos de implementación y exponer solo lo necesario. | Las clases del sistema encapsulan sus datos y comportamientos, exponiendo solo métodos públicos necesarios. Por ejemplo, la clase Vehículo encapsula todos los atributos relacionados con un vehículo y solo expone métodos para  operaciones permitidas. |
| **Modularidad** | División del sistema en componentes funcionales independientes. | El sistema se ha dividido en módulos independientes (gestión de salida, reportes, administración) que pueden ser desarrollados, probados y  mantenidos por separado. |
| **Abstracción** | Simplificar sistemas complejos enfocándose en lo esencial y ocultando detalles no relevantes. | Se utilizan interfaces y clases abstractas para definir comportamientos generales, permitiendo implementaciones específicas sin afectar a los  consumidores de estas interfaces. |
| **Principio de responsabilidad única (SRP)** | Una clase debe tener una única razón para cambiar. | Cada controlador y servicio en el sistema tiene una única responsabilidad, como, por ejemplo, el controlador de salida de vehículos solo maneja la lógica relacionada con registrar nuevos  ingresos. |
| **Principio abierto/cerrado (OCP)** | Las entidades deben estar abiertas para extensión, pero cerradas para modificación. | El sistema permite añadir nuevos tipos de vehículos o nuevas reglas de negocio sin modificar el código existente, mediante el uso de patrones de diseño como Strategy y  Factory. |

| **Principio de sustitución de Liskov (LSP)** | Los objetos de una superclase deben poder ser reemplazados por objetos de subclases sin afectar la  funcionalidad. | Las implementaciones específicas de interfaces (como diferentes tipos de vehículos) pueden ser utilizadas indistintamente donde se espera la  interfaz base. |
| --- | --- | --- |
| **Principio de segregación de interfaces (ISP)** | Los clientes no deben verse forzados a depender de interfaces que no utilizan. | Las interfaces del sistema están divididas según su propósito, evitando interfaces monolíticas. Por ejemplo, existen interfaces separadas para gestión de usuarios, gestión de  vehículos y generación de reportes. |
| **Principio de inversión de dependencias (DIP)** | Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones. | Se implementa inyección de dependencias para desacoplar componentes, permitiendo que los servicios de alto nivel dependan de abstracciones en lugar de  implementaciones concretas. |

* 1. **Diseño centrado en el usuario (UX/UI, prototipos, experiencia de usuario)**

El sistema se ha desarrollado siguiendo principios de diseño centrado en el usuario, con el objetivo de proporcionar una experiencia intuitiva y eficiente para los operadores que gestionan la salida de vehículos. Los aspectos clave del diseño centrado en el usuario incluyen:

## Investigación y análisis de usuarios

Se realizaron entrevistas y observaciones con operadores de estacionamientos y usuarios del sistema para identificar sus necesidades, puntos de dolor y flujos de trabajo. Este análisis inicial permitió definir perfiles de usuario y escenarios de uso que guiaron las decisiones de diseño posteriores.

## Arquitectura de información

La estructura de navegación y organización de la información se diseñó para reflejar los modelos mentales de los usuarios, con categorías intuitivas y una jerarquía clara que facilita encontrar las funciones necesarias en el momento adecuado.

## Prototipado iterativo

Se desarrollaron prototipos de baja y alta fidelidad para validar conceptos de diseño. Los prototipos evolucionaron a través de múltiples iteraciones basadas en pruebas de usabilidad con usuarios reales, permitiendo identificar y resolver problemas antes de la implementación final.

## Principios de diseño visual

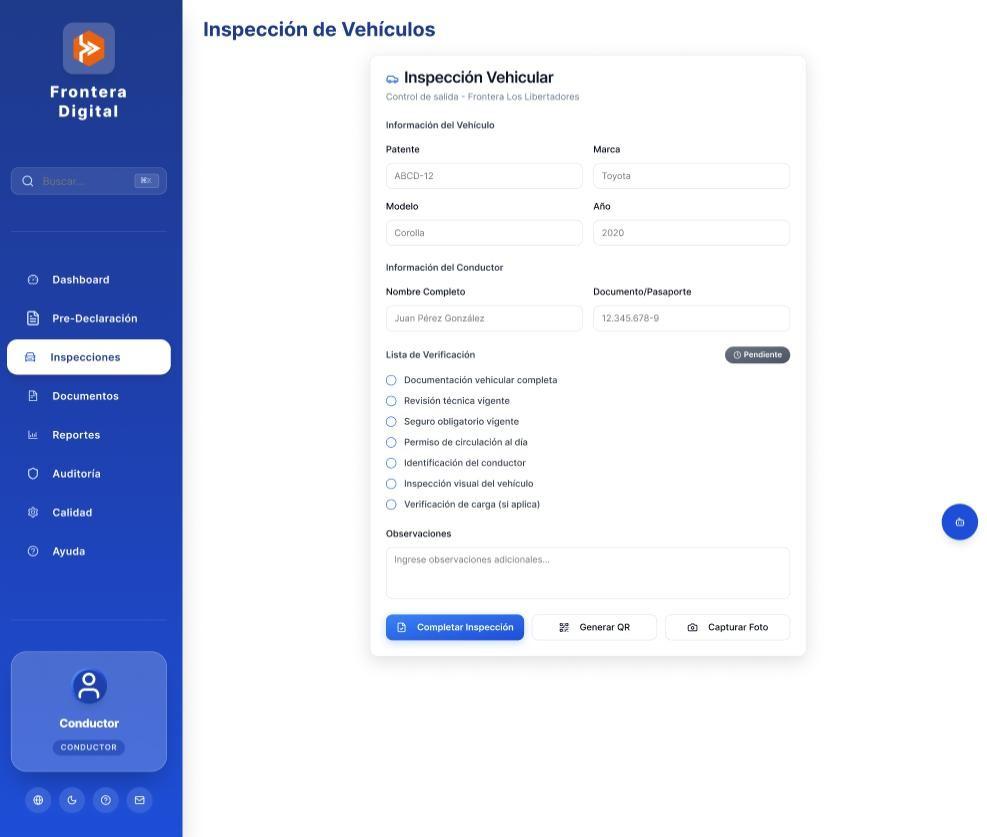
La interfaz implementa principios de diseño visual como:

* + - * **Consistencia:** Uso coherente de elementos visuales, terminología y patrones de interacción en toda la aplicación.
      * **Jerarquía visual:** Elementos organizados según su importancia para guiar la atención del usuario.
      * **Retroalimentación:** Respuestas visuales claras a las acciones del usuario para confirmar operaciones y transmitir el estado del sistema.
      * **Accesibilidad:** Conformidad con estándares WCAG para garantizar que el sistema sea utilizable por personas con diversas capacidades.

## Evaluación y mejora continua

El diseño UX/UI no se considera un proceso terminado sino un ciclo continuo de mejora. Se implementan métricas de usabilidad y satisfacción de usuario que se monitorean regularmente para identificar oportunidades de optimización.

## Prototipo



**7.1. Descripción general del prototipo**

El prototipo del Sistema de Salida de Vehículos Chile-Argentina representa una implementación visual interactiva de las principales funcionalidades del sistema. Este prototipo fue desarrollado para validar la experiencia de usuario y demostrar los flujos de trabajo críticos antes de la implementación final. El prototipo incluye tres pantallas principales que cubren el flujo completo del proceso de salida vehicular:

**- Registro inicial de la solicitud de salida**

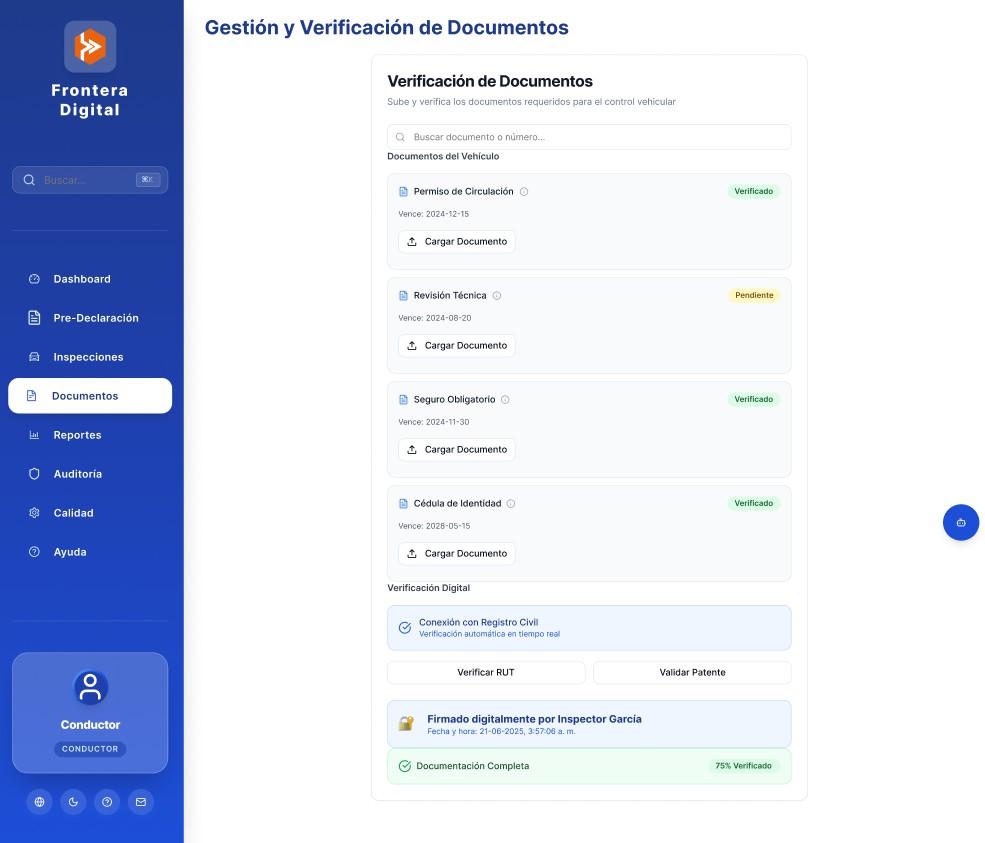
**- Verificación de documentación por parte del inspector**

**- Dashboard administrativo para supervisión y reportes**

Cada pantalla fue diseñada siguiendo principios de usabilidad y accesibilidad, con especial énfasis en la eficiencia operativa requerida en el entorno fronterizo.

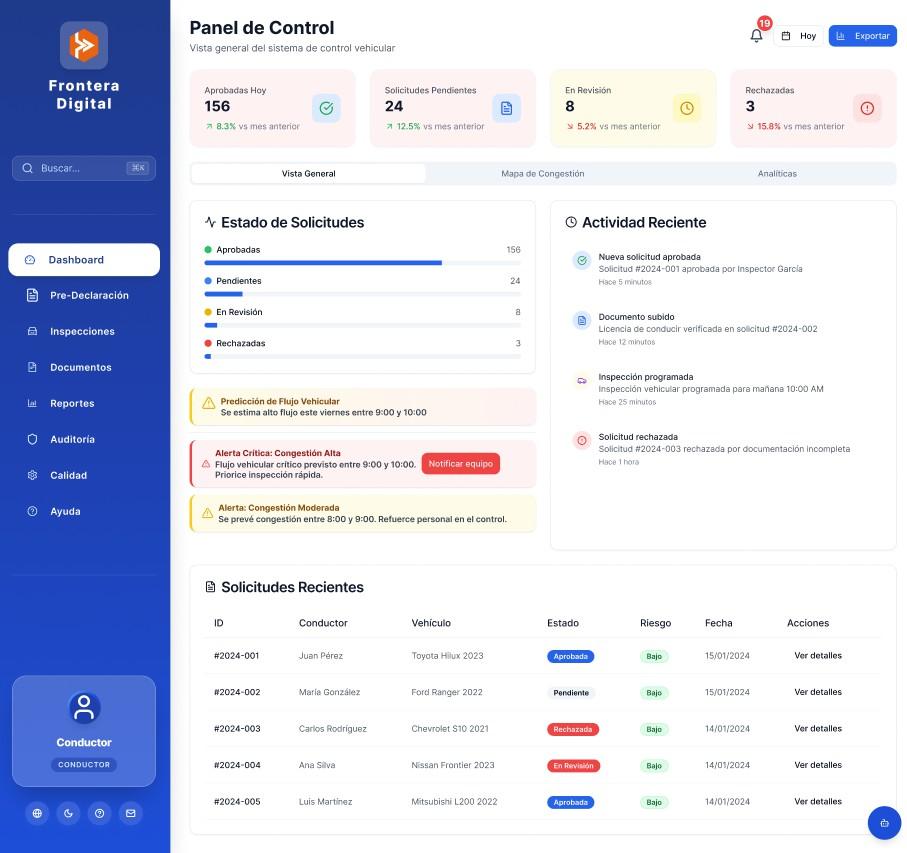
**7.2.Pantalla de registro de salida vehicular**

* Formulario con campo para placa (ocr/scan) + datos del conductor y vehículo.
* Ilustra el paso inicial crítico del flujo, donde parte la validación.



## Pantalla de verificación de documentos

* Visualización de cédula, licencia, revisión técnica, con botones de “Aprobar / Rechazar” y alertas si hay vencimientos o inconsistencias.
* Refleja la funcionalidad central de apoyar al inspector en decisiones quick-approve.



## Dashboard administrativo

* Estadísticas diarias, gráfico de flujo vehicular y alertas por irregularidades o reportes.
* Es vital para supervisores, muestra el resultado analítico del uso del sistema.

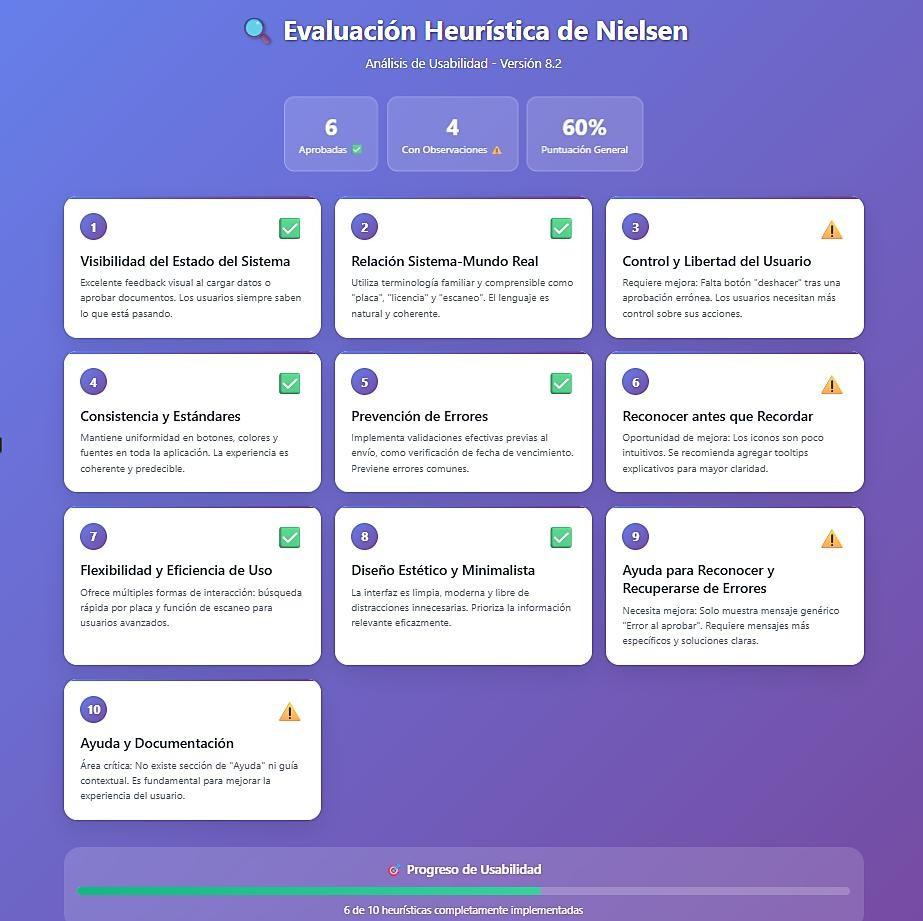
## 7.3. Herramientas de prototipado utilizadas Uso de Figma:

* **Diseño colaborativo en tiempo real:** permitió que todos los integrantes del equipo participaran en la creación y revisión del prototipo simultáneamente.
* **Prototipado interactivo completo:** se construyeron pantallas navegables para simular el flujo completo del sistema de salida vehicular, desde el ingreso de placa hasta la generación de reportes.
* **Estilos y componentes reutilizables:** se definieron botones, formularios, tarjetas de datos y modales como *components* reutilizables, lo cual asegura consistencia visual en toda la interfaz.
* **Versionado automático:** Figma guarda historial de versiones en la nube, facilitando revertir cambios y comparar iteraciones del diseño.
* **Responsividad simulada:** se diseñaron pantallas que adaptan su contenido a distintos tamaños (monitores táctiles en frontera, tablets, etc.).

## Plugins utilizados dentro de Figma:

* + **Iconify** para importar íconos estándar.
  + **Color Palettes** para probar esquemas de color accesibles.
  + **Wireframe** para primeros bosquejos rápidos antes del diseño final.

# EVALUACIÓN DE CALIDAD HEURÍSTICA DE NIELSEN



## 8.1 Metodología de evaluación La evaluación heurística se realizó utilizando los 10 principios de usabilidad de Jakob Nielsen, aplicados específicamente al contexto de un sistema de control fronterizo. Se evaluaron las tres pantallas principales del prototipo mediante la siguiente metodología:

## Evaluación individual por cada integrante del equipo

## Identificación de problemas de usabilidad por pantalla

## Clasificación de problemas según severidad (1-4)

## Consolidación de resultados y propuestas de mejora

## 8.2 Criterios de evaluación aplicados Los criterios evaluados fueron:

## Visibilidad del estado del sistema

## Coincidencia entre el sistema y el mundo real

## Control y libertad del usuario

## Consistencia y estándares

## Prevención de errores

## Reconocimiento antes que recordación

## Flexibilidad y eficiencia de uso

## Estética y diseño minimalista

## Ayuda para reconocer, diagnosticar y recuperarse de errores

## Ayuda y documentación

## Cada criterio fue evaluado en escala de 1-4 donde:

## 1: No es un problema

## 2: Problema cosmético

## 3: Problema menor

## 4: Problema mayor

## 8.3 Análisis y métricas de resultados

* **Fortalezas**: visibilidad, prevención de errores, consistencia, estética.
* **Áreas de mejora**: Clarificación de iconos, mensajes de error útiles, sección de ayuda.

## Métricas propuestas:

* + **Tasa de errores evitados**: medir cuántos rechazos antes de envío.
  + **Índice de correcciones**: cuántas veces se usa “deshacer” o se corrige

aprobaciones.

* + **Tiempo de aprendizaje**: medir cuánto tarda un nuevo inspector en completar un caso sin ayuda.

1. **Control de versiones**

## Propósito

Asegurar trazabilidad de cambios, colaboraciones ordenadas, y control sobre qué versión se está desplegando en producción.

## Control de versión utilizado

* + - **Esquema semántico (SemVer)**:
      * MAJOR.MINOR. PATCH
      * Ej: v1.2.0: cambios menores añadidos sin romper compatibilidad
      * v2.0.0: cambios que modifican estructura o flujos.
      * **Justificación**: Ofrece claridad para desarrollos iterativos, compatibilidad y colaboración segura.

## Herramientas de versionamiento

Herramientas utilizadas:

Se utilizó Git como sistema de control de versiones local y GitHub como plataforma remota para sincronización y colaboración.

El repositorio implementa ramas como main, develop y feature/nueva-funcion. También se usaron pull requests para control de cambios y revisiones por el equipo.

# CONCLUSIONES

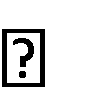
La arquitectura presentada en este documento para el sistema de gestión de salida de vehículos proporciona una base sólida para el desarrollo de una solución robusta, escalable y mantenible. A continuación, se presentan las principales conclusiones derivadas del diseño arquitectónico:

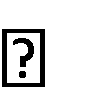
1. **Alineación con objetivos de negocio:** La arquitectura propuesta satisface los requerimientos funcionales y no funcionales del sistema, permitiendo una gestión eficiente de la salida de vehículos, lo que contribuye directamente a los objetivos operativos de la organización.
2. **Flexibilidad y extensibilidad:** El uso de patrones arquitectónicos y principios de diseño sólidos proporciona la flexibilidad necesaria para adaptarse a cambios futuros en los requerimientos del negocio. La estructura modular facilita la incorporación de nuevas funcionalidades sin afectar los componentes existentes.
3. **Rendimiento y escalabilidad:** Las decisiones arquitectónicas relacionadas con la distribución de componentes y la gestión de recursos permiten que el sistema maneje adecuadamente tanto el volumen actual de operaciones como el crecimiento proyectado, garantizando tiempos de respuesta óptimos incluso en períodos de alta demanda.
4. **Mantenibilidad:** La separación clara de responsabilidades, la cohesión alta y el acoplamiento bajo entre componentes facilitan las actividades de mantenimiento y evolución del sistema, reduciendo el costo total de propiedad a largo plazo.
5. **Experiencia de usuario:** El enfoque centrado en el usuario ha permitido desarrollar una interfaz intuitiva y eficiente que optimiza los flujos de trabajo de los operadores, reduciendo errores y aumentando la productividad.
6. **Seguridad y confiabilidad:** La arquitectura implementa mecanismos de seguridad en múltiples niveles y estrategias de tolerancia a fallos para garantizar la disponibilidad y protección de los datos, aspectos críticos en un sistema que gestiona activos físicos como vehículos.
7. **Integración:** La arquitectura facilita la integración con sistemas externos como servicios de pago, sistemas de identificación vehicular y plataformas de seguridad, permitiendo una operación fluida dentro del ecosistema tecnológico de la organización.

# 10.1. BIBLIOGRAFÍA

## Software Architecture in Practice

https://es.ebook.de/de/product/3814405/arquitectura\_de\_software\_en\_la\_practica.html

**Documenting Software Architectures: Views and Beyond** https://[www.amazon.es/Documenting-Software-Architectures-Second-](http://www.amazon.es/Documenting-Software-Architectures-Second-) Viewpoints/dp/0321552687

**Designing Software Architectures: A Practical Approach** https://[www.amazon.es/Designing-Software-Architectures-Practical-](http://www.amazon.es/Designing-Software-Architectures-Practical-) Approach/dp/0134390784

## The Unified Modeling Language User Guide

https://[www.amazon.es/Unified-Modeling-Language-User-Guide/dp/0321267974](http://www.amazon.es/Unified-Modeling-Language-User-Guide/dp/0321267974)

## Modelo de vistas 4+1 de Kruchten

https://[www.arquitecturadesoftware.org/modelo-4-1/](http://www.arquitecturadesoftware.org/modelo-4-1/)

## Guía UML: tipos de diagramas y usos

https://[www.genbeta.com/desarrollo/que-es-uml-tipos-diagramas](http://www.genbeta.com/desarrollo/que-es-uml-tipos-diagramas)

## Guía oficial de PlantUML (español)

https://plantuml.com/es/guide

## Patrones de arquitectura para microservicios

https://apiumhub.com/es/tech-blog-barcelona/patrones-arquitectura-microservicios/

## Diseño de APIs REST

https://[www.redhat.com/es/topics/api/what-is-a-rest-api](http://www.redhat.com/es/topics/api/what-is-a-rest-api)

## Autenticación y seguridad con JWT y OAuth 2.0

https://auth0.com/es/docs

## Accesibilidad Web: WCAG 2.1 en español

<https://www.w3.org/WAI/WCAG21/es>

## Buenas prácticas con PostgreSQL

<https://www.postgresql.org/docs/current/index.html>