

Prototipo de sistema descentralizado para la gestión y verificación de evidencias digitales en fotocomparendos aplicando Blockchain e IPFS

Laura Catalina Preciado Ballen

Cristian Stiven Guzman Tovar

Proyecto de Monografía para Optar por el Título de
Ingeniero(a) de Sistemas



Universidad Distrital Francisco José De Caldas

Facultad de Ingeniería

Colombia, Bogotá D.C.

Julio de 2025

Índice

Introducción	1
Formulación del problema	1
Objetivos	5
Impacto o alcance esperado	6
Justificación	10
Pertinencia social, tecnológica y legal	13
Originalidad e innovación	13
Impacto esperado	13
Relación con los objetivos del proyecto	14
Marco teórico	15
El paradigma de la confianza descentralizada	15
Fundamentos de los sistemas distribuidos y redes descentralizadas	15
Tecnologías para la gestión descentralizada de evidencia	16
Blockchain: un registro distribuido, inmutable y transparente	16
IPFS: almacenamiento verificable mediante direccionamiento por contenido	18
Arquitectura de la solución: sinergia blockchain-IPFS con el transacción off-chain	18
Fundamentos criptográficos aplicados	19
Estado del arte	20
Blockchain para registros gubernamentales y gestión de sanciones	20
Integración de blockchain e IPFS para datos voluminosos y verificables	21
Gestión de evidencia digital y cadena de custodia con DLT	22
Implementaciones reales de blockchain en gobiernos	24
Registro de propiedad en Suecia	24
Integridad de registros clínicos en Estonia	24

Síntesis y relevancia para el proyecto	24
Funcionamiento de los fotocomparendos en Bogotá: mecanismos, regulación e impacto	25
Aplicaciones específicas en gestión de tráfico e infracciones	27
Avances significativos	31
Limitaciones identificadas	31
Novedad y relevancia del prototipo	33
La relevancia del prototipo se justifica por su potencial para:	33
Análisis de tendencia internacional	33
Alcance	38
Enfoque y delimitación geográfica	38
Componentes del prototipo	38
Fuera del alcance	39
Entregables	39
Criterios de éxito	39
Limitaciones del prototipo	39
Entorno de validación	40
Integración y comparación con sistemas existentes	40
Aspectos técnicos y de escalabilidad	41
Seguridad y robustez	41
Metodología	44
Metodología de investigación	44
Metodología de desarrollo de software: enfoque por prototipos	44
Introducción a los artefactos técnicos del diseño	45
Diseño del prototipo	46
Definición de requisitos	46

Diagrama de casos de uso del sistema de gestión de infracciones de tránsito	48
Diagrama de despliegue	48
Diagrama de clases	51
Diagrama de actividades	52
Interfaz de usuario	52
Plan de pruebas	62
Introducción y propósito	62
Alcance de las pruebas	62
Fuera de alcance	62
Entorno de pruebas (simulación controlada)	63
Tipos de pruebas y casos de prueba detallados	65
Pruebas de inmutabilidad	65
Pruebas de rendimiento básico	66
Casos de prueba de inmutabilidad y verificabilidad	68
Estrategia de pruebas del frontend	68
Herramientas y tecnologías	69
Pruebas unitarias	69
Pruebas de integración	69
Resultados de las pruebas de inmutabilidad y verificabilidad del prototipo	70
Pruebas de inmutabilidad en blockchain	70
Verificación de integridad con IPFS	70
Verificabilidad transparente del registro	70
Casos de prueba funcionales	71
Casos de prueba de inmutabilidad	72
Pruebas de Rendimiento Básico	72
Cumplimiento de objetivos específicos	72

Resultados detallados de pruebas backend	73
Implementación del prototipo	78
Entorno de desarrollo y herramientas	78
Stack tecnológico implementado	78
Tecnologías backend	78
Tecnologías blockchain	79
Almacenamiento descentralizado	80
Tecnologías frontend	80
Frameworks de testing	81
Implementación de la capa pública: Ethereum	81
Desarrollo del smart contract	81
Despliegue y configuración	84
Implementación de la capa privada: Hyperledger Fabric	86
Configuración de la red	86
Desarrollo del chaincode	86
Servicio de sincronización entre blockchains	87
Arquitectura del servicio	87
Flujo de sincronización	88
Manejo de errores y reintentos	88
Implementación de IPFS dual	89
IPFS privado	89
IPFS público	89
Desarrollo del backend: API REST	90
Arquitectura de servicios	90
Endpoints principales	90
Middleware de seguridad	90
Documentación con Swagger	91

Desarrollo del frontend	92
Arquitectura de componentes	92
Gestión de estado	93
Interacción con backend	93
Integración con sistemas externos	94
Simulación de APIs gubernamentales	94
Consideraciones para integración real	94
Desafíos técnicos y soluciones implementadas	95
Compatibilidad de Módulos ESM	95
Optimización de Gas en Ethereum	95
Sincronización Asíncrona	95
Manejo de Archivos Grandes en IPFS	95
Testing y validación	96
Tests de Smart Contracts	96
Tests de Backend	96
Tests de Frontend	96
Discusión y análisis	98
Cumplimiento de objetivos	98
Objetivo general	98
Objetivo específico 1: implementar blockchain para inmutabilidad	98
Objetivo específico 2: almacenamiento descentralizado	99
Objetivo específico 3: API REST funcional	99
Objetivo específico 4: interfaz de usuario	100
Análisis de la arquitectura híbrida	100
Ventajas sobre arquitecturas monolíticas	100
Comparación con el sistema actual (FÉNIX)	101
Comparación con estado del arte	101

Implicaciones del trabajo	103
Impacto técnico	103
Impacto social	104
Impacto económico	104
Limitaciones y restricciones	105
Limitaciones Técnicas del Prototipo	105
Limitaciones Metodológicas	106
Limitaciones de Seguridad	106
Lecciones aprendidas	107
Complejidad de Hyperledger Fabric	107
Trade-off Descentralización vs Rendimiento	107
Importancia de UX en Sistemas Blockchain	108
Diseño Modular para Evolución	108
Consideraciones para despliegue en producción	108
Infraestructura	108
Seguridad	109
Operaciones	109
Conclusiones y trabajo futuro	110
Conclusiones generales	110
Conclusiones por objetivo específico	110
Objetivo 1: blockchain para inmutabilidad	110
Objetivo 2: almacenamiento descentralizado con IPFS	111
Objetivo 3: API REST para integración	111
Objetivo 4: interfaz de usuario intuitiva	112
Contribuciones del trabajo	112
Contribución académica	112
Contribución técnica	113

Contribución social	113
Recomendaciones	114
Para implementación en producción	114
Para adopción institucional	114
Para replicación en otros municipios	115
Trabajo futuro	115
Fase 1: Completar Arquitectura Híbrida (3-6 meses)	115
Fase 2: Despliegue en Servidor Universitario (1-2 meses)	116
Fase 3: Piloto Controlado con SDM (6 meses)	116
Fase 4: Funcionalidades Avanzadas (6-12 meses)	116
Fase 5: Escalamiento Nacional (12+ meses)	117
Reflexiones finales	117
Anexos	119
Anexo A: código fuente relevante	119
Smart Contract FineManagement.sol (Extracto Comentado)	119
Servicio Backend: FineService.ts (Extracto)	127
Servicio IPFS: IPFSService.ts (Extracto)	130
Anexo B: configuración de infraestructura	132
Configuración de Hardhat (hardhat.config.cjs)	132
Configuración de Docker Compose para Hyperledger Fabric	133
Anexo C: manual de instalación	135
Prerrequisitos	135
Paso 1: Instalación de Node.js	136
Paso 2: Instalación de IPFS	136
Paso 3: Clonar e Instalar el Proyecto	137
Paso 4: Configurar Variables de Entorno	137
Paso 5: Compilar y Desplegar Smart Contracts	138

Paso 6: Iniciar el Backend	139
Paso 7: Iniciar el Frontend	139
Verificación de la Instalación	139
Anexo D: manual de usuario	140
Manual para Agentes de Tránsito	140
Manual para ciudadanos	141
Anexo E: glosario de términos	142

Índice de figuras

Figura 1. Estadísticas de comparendos emitidos en Bogotá entre enero de 2018 y agosto de 2024	2
Figura 2. Distribución global de la producción científica sobre blockchain y gestión de infracciones	34
Figura 3. Evolución anual de publicaciones en los países líderes del tema (Brasil, México, Colombia, España y Perú)	35
Figura 4. Nube de palabras de los términos más recurrentes en la literatura sobre blockchain e infracciones	36
Figura 5. Mapa temático de los principales temas de investigación en el área . . .	37
Figura 6. Diagrama de casos de uso del sistema de gestión de infracciones de tránsito	49
Figura 7. Diagrama de despliegue de la arquitectura del sistema	50
Figura 8. Diagrama de clases del sistema de gestión de multas	53
Figura 9. Diagrama de actividades para el proceso de apelación de multa	54
Figura 10. Diagrama de actividades para el proceso de creación de multa	55
Figura 11. Pantalla de login del sistema	56
Figura 12. Pantalla de recuperación de contraseña	57
Figura 13. Dashboard del agente de tránsito	58
Figura 14. Pantalla de consulta del estado de multa	59
Figura 15. Pantalla de consulta de detalle de multa	60
Figura 16. Pantalla de consulta de multas para propietarios de vehículos	61
17. Panel de Agente de Tránsito - Registro de Multa	92
18. Panel Ciudadano - Consulta y Verificación de Multas	93
19. Dashboard Administrativo - Estadísticas y Métricas	94
20. Pantalla de Registro de Multa - Panel del Agente	141
21. Pantalla de Consulta Pública - Panel Ciudadano	142

Índice de tablas

Tabla 1. Comparación entre bases de datos tradicionales y blockchain para gestión de registros gubernamentales	4
2. Comparación entre un modelo centralizado y un modelo descentralizado . . .	11
Tabla 2. Comparación entre un modelo centralizado y un modelo descentralizado .	11
3. Análisis Comparativo del Estado del Arte en Gestión de Infracciones con Blockchain	28
Tabla 3. Análisis Comparativo del Estado del Arte en Gestión de Infracciones con Blockchain	28
Tabla 2. Casos de prueba funcionales para validar operaciones básicas del sistema	64
Tabla 3. Casos de prueba de inmutabilidad para validar resistencia a modificaciones	65
Tabla 4. Resultados de pruebas de inmutabilidad del sistema	66
Tabla 5. Resultados de pruebas funcionales del sistema	71
Tabla 6. Resumen de casos de prueba de inmutabilidad ejecutados	72
Tabla 7. Tiempos promedio de operaciones en el entorno de prueba	73
10. Relación entre objetivos específicos, técnicas de validación y resultados . . .	74
11. Resultados de pruebas del backend por módulo	75
12. Validaciones de seguridad implementadas y verificadas	77
13. Endpoints principales de la API REST	91
14. Comparación Sistema FÉNIX vs Arquitectura Híbrida	102
15. Glosario de Términos Técnicos	142

Introducción

En Colombia, la gestión de fotocomparendos ha sido objeto de controversia debido a fallas en la transparencia y posibles manipulaciones en el proceso de registro y validación de infracciones. La falta de un sistema confiable ha generado desconfianza entre los ciudadanos, lo que evidencia la necesidad de una solución que garantice la integridad, inmutabilidad y verificabilidad de la información.

La tecnología Blockchain ha demostrado ser una alternativa eficaz para el almacenamiento seguro y descentralizado de datos, asegurando que una vez registrados, estos no puedan ser alterados sin dejar rastro. A través de contratos inteligentes, es posible automatizar la validación y el procesamiento de fotocomparendos, reduciendo la intervención humana y minimizando el riesgo de corrupción o errores administrativos.

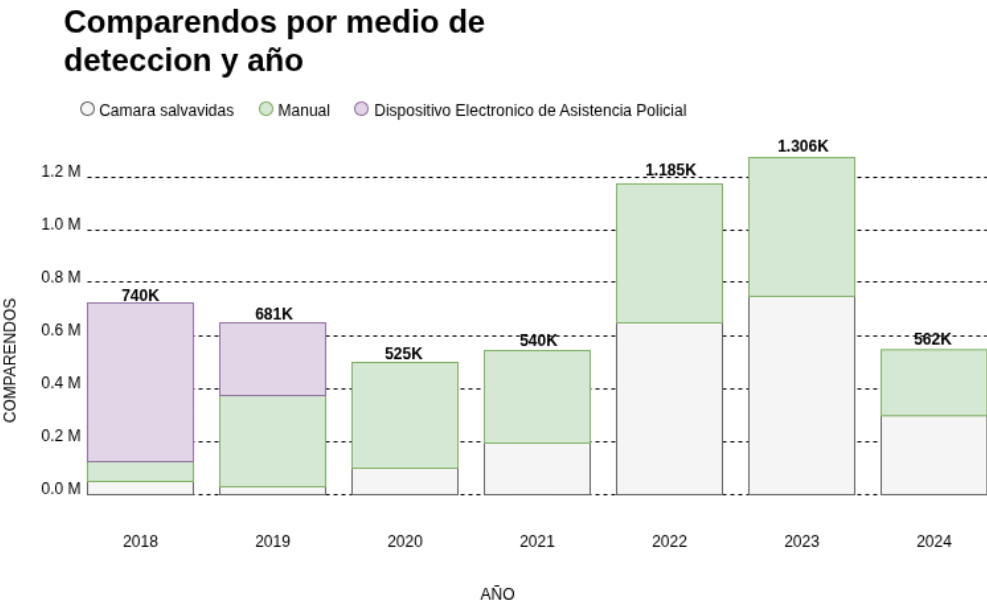
Este trabajo propone el diseño e implementación de un prototipo basado en Blockchain para la gestión de fotocomparendos en Bogotá, con el objetivo de garantizar la transparencia del proceso. Se utilizarán contratos inteligentes para registrar cada infracción, permitiendo que cualquier actor autorizado pueda verificar su autenticidad sin necesidad de intermediarios. Mediante pruebas y simulaciones, se evaluará la viabilidad del sistema, demostrando cómo esta tecnología puede fortalecer la confianza en los procesos de control de tránsito y mejorar la eficiencia en la gestión de sanciones.

Formulación del problema

La gestión de comparendos en Bogotá es un proceso de gran escala. Según datos del Observatorio de Movilidad, entre enero de 2018 y agosto de 2024 se emitieron más de 1.9 millones de comparendos a través de cámaras salvavidas, evidenciando la importancia sistémica de este proceso para la regulación del tránsito en la ciudad, como se presenta en la Figura 1 se observa los diferentes métodos utilizados para crear los comparendos. Esta operación se apoya en el sistema FÉNIX, una aplicación con infraestructura en la nube, cuya arquitectura de datos y control de acceso opera bajo un paradigma centralizado. En el sistema actual, la validez e inmutabilidad de los registros de infracciones se

Figura 1

Estadísticas de comparendos emitidos en Bogotá entre enero de 2018 y agosto de 2024



Nota. Elaboración propia basado en datos del Observatorio de Movilidad.

fundamenta en los procedimientos administrativos y en la gestión de los funcionarios responsables del sistema (Consejo de Estado, 2018). Los cambios en la información solo pueden ser detectados por las entidades autorizadas, lo que implica que el control sobre los registros depende directamente de la correcta aplicación de las políticas internas y del seguimiento realizado por dichas entidades (Consejo de Estado, 2019). La evidencia generada se conserva bajo un modelo centralizado, en el cual la confianza en la integridad de los datos se sostiene en mecanismos administrativos y controles internos, más que en garantías técnicas accesibles públicamente (Departamento Administrativo de la Función Pública, 2021). La potestad sancionatoria y el debido procedimiento administrativo aseguran la validez de los actos administrativos y la correcta motivación en la imposición

de sanciones (Corte Constitucional, 2022; Gamero Casado y Fernández Ramos, s.f.).

De acuerdo con la Auditoría de Cumplimiento de la Contraloría de Bogotá (2024), en el proceso de desarrollo del sistema FÉNIX se identificaron dificultades relacionadas con la supervisión contractual, lo que derivó en retrasos, duplicidad de sistemas y un presunto detrimento patrimonial estimado en más de \$8.000 millones de pesos. Estos hallazgos reflejan que, desde su implementación, la plataforma ha enfrentado retos significativos en materia de gobernanza y gestión, los cuales han tenido impacto en la eficiencia administrativa y en la sostenibilidad financiera del proyecto.

Estas debilidades se manifiestan en la operación técnica actual. A nivel operativo, el riesgo de integridad se materializa en una fricción a gran escala con la ciudadanía. Un análisis correlacional de fuentes oficiales para el primer semestre de 2025 revela la magnitud de esta fricción: frente a 457.000 comparendos impuestos [Observatorio de Movilidad, 2025], se gestionaron 155.854 PQRSD [Informe de Gestión PQRSD, 2025]. De estos datos se deduce una Tasa de Impugnación general del 34.1 %, un indicador cuantitativo que sugiere que al menos uno de cada tres actos administrativos del sistema genera una disputa formal, reflejando una carga administrativa insostenible y un déficit de confianza.

La desconfianza generada por estas opacidades y dificultades procesales crea un vacío que es explotado por terceros, afectando directamente al ciudadano. Reportajes de prensa documentan cómo la ausencia de canales oficiales percibidos como confiables ha fomentado la aparición de redes de fraude, como el caso de Juzto.co, donde miles de ciudadanos fueron estafados con promesas de impugnaciones garantizadas, resultando en trámites inconclusos y mayores deudas (Semana, 2023).

La identificación de estas limitaciones permite estructurar el problema en torno a variables que reflejan tanto el modelo de confianza actual como sus impactos técnicos, operativos y financieros. La Tabla 1 sintetiza estas variables y los indicadores asociados, mostrando cómo el paradigma centralizado de gestión condiciona la integridad de los datos, la eficiencia administrativa, la confianza ciudadana y la sostenibilidad del sistema.

Tabla 1*Comparación entre bases de bd y blockchain para gestión de registros gubernamentales*

Característica	Base de Datos Convencional	Blockchain
Modelo de confianza	Se basa en un administrador central (entidad de TI)	Confianza distribuida entre múltiples nodos
Inmutabilidad	Registros pueden ser modificados o eliminados por administradores	Los registros son inmutables por diseño
Trazabilidad / Auditoría	Depende de la implementación y control interno	Historial completo e inalterable disponible
Riesgo de corrupción interna	Alto, si hay privilegios indebidos o colusión	Bajo, no se puede alterar sin consenso de la red
Seguridad criptográfica	Opcional, no siempre integrada nativamente	Integrada (firmas digitales, hashes, cifrado)
Disponibilidad / tolerancia a fallos	Riesgo de puntos únicos de falla	Alta disponibilidad por replicación descentralizada
Velocidad de operación	Alta velocidad en lectura/escritura	Menor velocidad, prioriza integridad y consenso

Nota. Elaboración propia.

En síntesis, el problema se formula como un Riesgo de Integridad de Datos inherente al paradigma de confianza centralizada del sistema de fotocomparendos. Este riesgo se encuentra documentado por debilidades fundacionales en la gobernanza del proyecto y se manifiesta en consecuencias medibles: (i) una Tasa de Impugnación del 34.1 %; (ii) una carga operativa superior a 155 mil PQRSD semestrales; (iii) un presunto detrimento patrimonial por más de \$8.000 millones; y (iv) la vulnerabilidad de la ciudadanía a esquemas fraudulentos derivados de la falta de transparencia institucional.

Ante este panorama, surge la necesidad de explorar arquitecturas que permitan sustituir la confianza administrativa por garantías criptográficas. La pregunta central que guía este trabajo es:

¿Cómo mitigar el Riesgo de Integridad de Datos en el proceso de fotocomparendos en Bogotá?

Objetivos

Objetivo General. Desarrollar un prototipo para apoyar el registro y trazabilidad de estados en el proceso de fotocomparendos en Bogotá, aplicando tecnologías de redes distribuidas, con el fin de fortalecer la integridad, la autenticidad de la información, y reducir los riesgos asociados a su confidencialidad.

Objetivos específicos.

- Analizar el proceso actual de registro de fotocomparendos en Bogotá, a partir del marco jurídico y regulatorio que lo rige y de los informes de auditoría emitidos por la secretaria distrital de movilidad sobre la gestión de comparendos, para identificar requisitos funcionales, no funcionales y vulnerabilidades que el prototipo debe proporcionar.
- Desarrollar un prototipo con arquitectura híbrida fundamentada en la técnica de descomposición por confianza, integrando tecnologías de almacenamiento distribuido y contenido cifrado, asegurando que cada transacción incorpore los metadatos del

comparendo y disponiendo de una interfaz básica para demostrar que es posible un aplicativo transparente y confiable.

- Evaluar la viabilidad del prototipo desarrollado, mediante la ejecución de un plan de pruebas funcionales y evaluación de métricas de desempeño en un entorno de pruebas, para validar las condiciones de inmutabilidad, trazabilidad y seguridad.

Impacto o alcance esperado

Alcance. Enfoque y delimitación geográfica: Este trabajo se circunscribe al proceso de trazabilidad de estados de multas de tránsito automatizadas (fotomultas) emitidas por la Secretaría Distrital de Movilidad de Bogotá. Se excluyen de manera explícita los siguientes aspectos:

- Multas impuestas de forma presencial por agentes de tránsito.
- Procesos sancionatorios de otras ciudades o entidades territoriales.
- Funcionalidades de recaudo y pasarelas de pago (solo se registra el estado del pago, no se procesa el pago en sí).

Componentes del prototipo: El prototipo aborda los siguientes módulos funcionales:

- **Registro inmutable de la infracción:** Captura de metadatos (placa, fecha, hora, ubicación y tipo de infracción) y publicación del identificador de la evidencia en la blockchain (Hyperledger Fabric).
- **Almacenamiento descentralizado de evidencias:** Carga de la imagen o video de la fotomulta en IPFS y obtención de su hash.
- **Verificación pública:** Servicio de consulta que permite contrastar el hash guardado en la cadena con el archivo almacenado en IPFS.

- **Gestión del ciclo de vida de la multa:** Estados: Generada → Notificada → En apelación → Pagada → Cerrada. Cada transición queda registrada mediante eventos de contrato inteligente.
- **Interfaz mínima:** Panel Web para: (i) agentes que registran la infracción y (ii) ciudadanos que consultan la autenticidad y el estado de su fotomulta.

Fuera del alcance:

- Integración completa con sistemas legados del RUNT o SIMIT; se simula mediante datos de prueba.
- Implementación de un modelo económico (tarifas de gas, costos operativos reales).
- Implementación de algoritmos de detección automática de infracciones (visión por computador). Se parte de que la cámara ya detectó la infracción y generó la evidencia.
- Despliegue en un entorno de producción o capacidad más de 10 usuarios.

Entregables:

- Contrato inteligente en Solidity (o «chaincode» en Go, según la red seleccionada) con pruebas unitarias.
- Script de despliegue de red Hyperledger Fabric e instalación de IPFS local.
- Aplicación Web de demostración (frontend ligero) conectada a los servicios anteriores.
- Manual técnico que documenta la arquitectura y el flujo de datos.
- Informe de resultados de las pruebas funcionales y de rendimiento básico.

Criterios de éxito:

- Tiempo medio de publicación de una infracción ≤ 3 s en entorno de laboratorio.

- Coincidencia 100 % entre el hash almacenado en la cadena y la evidencia recuperada desde IPFS.
- Trazabilidad completa del historial de estados para al menos 50 multas de prueba.
- Ausencia de fallos críticos en pruebas de carga con 10 transacciones concurrentes.

Limitaciones del Prototipo. Es fundamental reconocer que, como prototipo desarrollado en un contexto académico, el presente estudio presenta ciertas limitaciones que definen el alcance de sus conclusiones y delinean claras oportunidades para futuras investigaciones. Las principales limitaciones son:

Entorno de Validación:

- **Validación en Entorno de Laboratorio:** El prototipo fue diseñado, desplegado y evaluado en un entorno de simulación controlado. No se sometió a pruebas en una infraestructura productiva real con la carga de transacciones y el volumen de usuarios que gestiona actualmente la Secretaría de Movilidad.
- **Uso de Datos Simulados:** Debido a estrictas normativas de privacidad y protección de datos personales que impiden el acceso a información real de ciudadanos y vehículos, todas las pruebas se realizaron con datos sintéticos.
- **Suposiciones sobre la Calidad de la Evidencia:** El sistema asume que las evidencias fotográficas (imágenes de fotocomparendos) son capturadas con una calidad suficiente para su procesamiento.

Integración y Comparación con Sistemas Existentes:

- **Integración Simulada con Sistemas Externos:** La interacción con plataformas gubernamentales clave como el RUNT y el SIMIT fue simulada a través de APIs de prueba (mocks).

- **Ausencia de Benchmarking Directo con el Sistema Actual (Fénix):** La falta de acceso al código fuente y a la arquitectura interna del sistema Fénix impidió realizar una comparación cuantitativa y directa.

Aspectos Técnicos y de Escalabilidad:

- **Proyección de Costos como Escenario de Referencia:** Los costos de infraestructura y desarrollo estimados corresponden a un escenario de referencia.
- **Estrategia de Persistencia en IPFS:** Para que la evidencia digital permanezca disponible a largo plazo en IPFS, es necesario que al menos un nodo la mantenga "pineada".

Seguridad y Robustez:

- **Ausencia de Pruebas de Seguridad Ofensivas:** El alcance del proyecto no incluyó la realización de auditorías de seguridad formales sobre los contratos inteligentes ni pruebas de penetración sobre la aplicación web.

Justificación

La gestión de registros públicos, como los fotocomparendos, en arquitecturas centralizadas presenta debilidades en materia de seguridad, transparencia y auditabilidad de la información. En el caso de Bogotá, el sistema FÉNIX sirve como un caso de estudio relevante. Auditorías oficiales de la Contraloría (**Informe 170100-0054-24**) (**Informe de Cumplimiento No. 90; 2023**) han documentado desafíos en su implementación y operación, incluyendo limitaciones en la integridad de los datos. Este escenario, sumado a la fricción operativa evidenciada por más de 153.000 PQRSD en un semestre, resalta la oportunidad de proponer nuevos modelos arquitectónicos que fortalezcan la confianza pública, que no dependa exclusivamente de la confianza en los procedimientos y administradores internos, pasando de un sistema donde la integridad se presume y audita a posteriori, a uno donde la integridad es una propiedad intrínseca, criptográficamente verificable desde su origen.

La finalidad de este proyecto no es proponer una modificación al sistema existente, sino diseñar y evaluar un prototipo autocontenido que demuestre un modelo de confianza fundamentalmente diferente. La propuesta busca, con el fin de evidenciar las diferencias estructurales entre el modelo convencional y el prototipo propuesto, en la Tabla se presenta una comparación detallada de sus características:

Tabla 2. *Comparación entre un modelo centralizado y un modelo descentralizado*

Característica	Modelo Centralizado	Modelo Descentralizado	Relevancia Contextual (Basado en el Caso de Estudio de la Auditoría No. 90)
Modelo de Confianza	Basado en la confianza en los administradores del sistema y en la robustez de los controles internos definidos.	Basado en un consenso criptográfico distribuido, donde la confianza reside en el protocolo y no en un intermediario.	La correcta asignación de roles es fundamental. La auditoría observó “ausencia de un profesional responsable de Seguridad de la Información” (págs. 20–25), subrayando la criticidad de los factores de gobernanza.
Integridad de Datos	La integridad se asegura mediante controles de acceso y logs de auditoría internos gestionados por la entidad.	La integridad es una propiedad intrínseca de la estructura de datos; los registros son inmutables por diseño.	La efectividad de los controles internos es fundamental. La auditoría documentó “Falta de control sobre la integridad y calidad de los datos migrados” (págs. 38–40) como punto de atención.

Cuadro 2 – continuación de la página anterior

Característica	Modelo Centralizado	Modelo Descentralizado	Relevancia Contextual (Basado en el Caso de Estudio de la Auditoría No. 90)
Gestión de Seguridad	Dependiente de políticas y procedimientos de seguridad definidos y ejecutados por la institución.	La seguridad es una propiedad inherente a la capa de protocolo, auditada de forma continua y global por la comunidad.	La formalización de procedimientos es clave. La auditoría identificó “falta de gestión formal de riesgos y controles” y “ausencia de un plan de seguridad para la infraestructura en la nube” (págs. 25–30).
Auditabilidad y Trazabilidad	La auditoría se realiza a través de logs internos, con acceso gestionado por la entidad y sujeto a sus políticas de retención y seguridad.	La traza de auditoría es transparente, inalterable por diseño y públicamente verificable por cualquier actor autorizado.	La consistencia de los registros internos es un factor de éxito. La auditoría observó “retrasos y baja velocidad de desarrollo” (págs. 15–20), subrayando la importancia de una gobernanza rigurosa.

Fuente: Elaboración propia, con hallazgos basados en la Auditoría de Cumplimiento No. 90 de la Contraloría de Bogotá D.C. (octubre de 2023) y la Auditoría de Cumplimiento 170100-0054-24.

Pertinencia social, tecnológica y legal

La pertinencia de este proyecto se enmarca en tres dimensiones:

- **Social y ciudadana:** Ofrece un modelo alternativo que responde a la necesidad de transparencia, permitiendo la verificación independiente y empoderando al ciudadano con herramientas de auditoría directa.
- **Tecnológica:** Demuestra cómo la integración de Blockchain (para registros inmutables) e IPFS (para evidencias con contenido direccionable) puede abordar los desafíos de seguridad y trazabilidad documentados en sistemas centralizados.
- **Legal e institucional:** El prototipo se alinea con los principios de eficiencia y transparencia exigidos por los organismos de control, sirviendo como un caso de estudio sobre cómo la tecnología puede fortalecer la rendición de cuentas.

Originalidad e innovación

La innovación de esta monografía radica en la concepción del prototipo como un laboratorio para un nuevo modelo de confianza. Mientras los sistemas tradicionales se centran en controles administrativos, esta propuesta explora un modelo distribuido y resistente a la manipulación por diseño. La DApp funciona como una prueba de concepto que integra inmutabilidad, gobernanza automatizada y almacenamiento descentralizado para demostrar una solución a una clase de problemas que las bases de datos centralizadas, por su naturaleza, no pueden resolver de manera nativa.

Impacto esperado

El impacto del proyecto se manifiesta en varias dimensiones:

- **Confianza por Diseño:** Muestra cómo la verificación independiente puede fortalecer la legitimidad de los procesos públicos.
- **Gobernanza Automatizada:** Ilustra cómo los contratos inteligentes pueden ejecutar reglas de negocio de forma predecible, reduciendo la dependencia de la

supervisión humana.

- **Escalabilidad en GovTech:** Constituye un caso de uso transferible a otros procesos que demandan alta integridad, sirviendo como un precedente para futuras innovaciones en la administración pública.

Relación con los objetivos del proyecto

Este prototipo responde a una problemática documentada en Bogotá y se inserta en la tendencia global de GovTech. Por ello, la adopción de blockchain en esta propuesta no es una preferencia, sino una respuesta técnica deliberada a los desafíos de integridad y confianza inherentes a los modelos centralizados, proponiendo una arquitectura donde la veracidad es una propiedad intrínseca y verificable del sistema.

Marco teórico

El marco conceptual y tecnologico que sustenta la propuesta del prototipo, presentan las teorías y modelos clave que justifican la selección de Blockchain e IPFS como componentes centrales, evidencian los principios inherentes de integridad, transparencia, resiliencia y auditabilidad en la gestión de evidencia digital crítica como los fotocomparendos.

El paradigma de la confianza descentralizada

Los sistemas de información tradicionales suelen depender de intermediarios centralizados o autoridades certificadoras para validar transacciones y garantizar la fiabilidad de los registros. La teoría de los modelos de confianza descentralizada, en cambio, analiza cómo establecer y mantener la confianza en entornos distribuidos donde tales autoridades centrales están ausentes (**swan2015blockchain**).

La relevancia de este modelo es fundamental para justificar el uso de la tecnología Blockchain en la gestión de fotocomparendos, ya que su propósito es precisamente reemplazar la necesidad de depositar confianza exclusiva en una única entidad para la custodia, validación e integridad de los registros. Blockchain habilita un cambio de paradigma: en lugar de confiar en un actor central, la confianza se distribuye y se deposita en la robustez del protocolo criptográfico subyacente (**nakamoto2008bitcoin**), en la transparencia de las reglas del sistema y en el consenso mayoritario de los participantes de la red (**antonopoulos2023mastering**). Este enfoque reduce drásticamente los puntos únicos de fallo y los vectores de corrupción asociados a la dependencia de intermediarios centralizados, quienes podrían ser comprometidos, cometer errores o actuar de manera malintencionada.

Fundamentos de los sistemas distribuidos y redes descentralizadas

El paradigma de la confianza descentralizada se sustenta en la teoría de los sistemas distribuidos, donde múltiples entidades autónomas, denominadas nodos, colaboran a través de una red para alcanzar un objetivo común, compartiendo tanto la carga computacional como el almacenamiento de datos (**vanSteen2017**). Estos sistemas se fundamentan en

principios como la distribución de recursos, la comunicación inter-nodo y mecanismos de coordinación que prescinden de intermediarios centrales (**coulouris2011**).

La relevancia de esta teoría para el presente proyecto es primordial, ya que tanto Blockchain como el InterPlanetary File System (IPFS) son implementaciones nativas de sistemas distribuidos. Su adopción conjunta promueve inherentemente:

- **Resiliencia:** Al eliminar puntos únicos de fallo (Single Points of Failure - SPOF).
- **Alta Disponibilidad:** Al permitir el acceso a datos y servicios desde múltiples nodos.
- **Resistencia a la Censura:** Dado que ninguna entidad individual posee control absoluto sobre la red o los datos almacenados (**antonopoulos2023mastering**).

Una característica esencial de estos sistemas es su arquitectura de red **Peer-to-Peer (P2P)**, donde los participantes se conectan y comparten recursos directamente entre sí, sin necesidad de un servidor central. En una red P2P, cada nodo puede actuar simultáneamente como cliente y servidor, lo que posibilita que el registro distribuido (ledger) se mantenga sincronizado y que los archivos puedan ser recuperados desde múltiples fuentes, garantizando la integridad de la información sin depender de una autoridad central.

Tecnologías para la gestión descentralizada de evidencia

Para materializar un sistema de gestión de fotocomparendos descentralizado, se requiere la sinergia de dos tipos de tecnologías: una para el registro inmutable de transacciones y otra para el almacenamiento verificable de la evidencia.

Blockchain: un registro distribuido, inmutable y transparente

Blockchain es un tipo específico de Tecnología de Ledger Distribuido (DLT), un sistema de registro digital caracterizado por ser distribuido, sincronizado y asegurado criptográficamente entre múltiples participantes (**narayanan2016bitcoin**). Su estructura fundamental se compone de **transacciones** —operaciones firmadas digitalmente que

modifican el estado del ledger de forma permanente (**antonopoulos2023mastering**)—agrupadas en bloques. Cada bloque contiene un hash criptográfico que lo vincula al anterior, formando una cadena cronológica e inmutable.

La **inmutabilidad** y la **transparencia** son los beneficios centrales que esta tecnología aporta (**swan2015blockchain**; **antonopoulos2023mastering**). La primera se logra mediante la estructura encadenada y los mecanismos de consenso distribuido (ej., Proof-of-Work (**nakamoto2008bitcoin**) o Proof-of-Stake (**king2012ppcoin**)), que hacen que la modificación de un bloque pasado sea computacionalmente prohibitiva. La segunda se habilita por la naturaleza replicada del ledger, permitiendo que actores autorizados puedan consultar y verificar la información de forma independiente. Dentro de este ecosistema, los **Smart Contracts** (Contratos Inteligentes) actúan como programas autoejecutables cuyo código define e impone automáticamente los términos de un proceso, permitiendo automatizar la gestión del ciclo de vida del comparendo (**szabo1997smart**; **wood2014ethereum**; **buterin2014next**).

Modelos arquitectónicos y elección para el prototipo. La tecnología Blockchain no es monolítica; existen diferentes arquitecturas:

- **Públicas (Permissionless):** Abiertas a cualquier participante, priorizan la descentralización radical (ej. Bitcoin, Ethereum) (**nakamoto2008bitcoin**).
- **Privadas:** Controladas por una única entidad, ofrecen alta eficiencia pero son centralizadas.
- **De Consorcio/Permisionadas (Permissioned):** Operadas por un grupo selecto de participantes autorizados. Ofrecen un equilibrio entre descentralización, rendimiento y confidencialidad, siendo la opción ideal para contextos gubernamentales y empresariales (**vukolic2015quest**; **cachin2018architecture**).

Para este prototipo, se opta por una **implementación permisionada** (simulada con Hyperledger Fabric), permitiendo que solo entidades autorizadas operen nodos y registren

transacciones, con un mecanismo de consenso eficiente (ej. Raft) adecuado para un sistema de gestión de registros.

IPFS: almacenamiento verificable mediante direccionamiento por contenido

Los ledgers de Blockchain no están optimizados para almacenar grandes volúmenes de datos (blobs), como las imágenes de los fotocomparendos (xu2019taxonomy). Para resolver esto, se utiliza un sistema de almacenamiento descentralizado. La elección de IPFS sobre alternativas centralizadas como AWS S3 es crucial para la integridad del sistema. Mientras que en un sistema centralizado el propietario puede modificar o eliminar unilateralmente un archivo (vogels2008eventually), IPFS opera bajo el paradigma del **direccionamiento por contenido (Content Addressing)** (benet2014ipfs; voigt2017gdpr).

En este modelo, la identidad única de un archivo, su Content Identifier (CID), es un **hash criptográfico** derivado directamente de su contenido. Esto establece un vínculo intrínseco e inmutable: si el contenido del archivo cambia, incluso mínimamente, su CID también cambiará. IPFS es un protocolo y red P2P que utiliza este principio: divide los archivos en bloques, calcula sus hashes y permite su recuperación a través de su CID, utilizando mecanismos como DHT para localizar los nodos que los poseen (maymounkov2002kademlia; benet2014ipfs).

Arquitectura de la solución: sinergia blockchain-IPFS con el transacción off-chain

La integración de ambas tecnologías se materializa mediante el patrón de almacenamiento **off-chain**. El flujo de trabajo es el siguiente:

1. La imagen probatoria del comparendo se carga a un nodo IPFS, obteniendo su CID único.
2. Se crea una transacción en la Blockchain (on-chain) que contiene este CID junto con los metadatos esenciales del comparendo (fecha, hora, lugar, placa).
3. Esta transacción se valida y registra de forma inmutable en el ledger.

Este enfoque crea un enlace criptográfico inalterable entre el registro oficial (en Blockchain) y la evidencia visual original (en IPFS). Cualquier intento de manipulación de la imagen almacenada en IPFS resultaría en un CID diferente, rompiendo explícitamente la cadena de custodia digital y haciendo que la alteración sea detectable de forma inmediata y algorítmica. La combinación de Blockchain e IPFS no solo sigue los principios de descentralización (**vanSteen2017**), sino que refuerza activamente los objetivos de inmutabilidad verificable y transparencia del sistema.

Fundamentos criptográficos aplicados

La criptografía proporciona los pilares matemáticos que garantizan la seguridad, integridad y autenticidad en todo el ecosistema del prototipo (**katz2020introduction**).

- **Funciones Hash Criptográficas:** Son algoritmos que transforman datos en una huella digital de tamaño fijo. Sus propiedades (unidireccionalidad, resistencia a colisiones, efecto avalancha) son vitales (**schneier2007applied; menezes1996handbook**). En este proyecto, se utilizan para: generar el CID en IPFS, asegurar la integridad de la cadena de bloques y crear identificadores únicos para las transacciones (**benet2014ipfs; nakamoto2008bitcoin**).
- **Criptografía Asimétrica y Firmas Digitales:** Basada en pares de claves (pública y privada), habilita las firmas digitales (**diffie2022new; rivest1978method**). Cuando un usuario autorizado registra un comparendo, utiliza su clave privada para firmar la transacción. Cualquier participante puede usar la clave pública correspondiente para verificar la firma, garantizando así la **autenticidad** y el **no repudio** de la acción (**katz2020introduction**).

Estado del arte

Blockchain para registros gubernamentales y gestión de sanciones

La aplicación de la tecnología Blockchain y DLT (Distributed Ledger Technology) en la administración pública ha sido un área de creciente interés, impulsada por las promesas teóricas de Inmutabilidad, Transparencia y Auditoría mejorada, fundamentales para la Confianza Descentralizada. La investigación sugiere que Blockchain puede transformar la gestión de registros oficiales, como licencias, títulos de propiedad y, potencialmente, multas o sanciones como los fotocomparendos.

Análisis de aplicación. La capacidad de crear un registro de Transacciones criptográficamente asegurado y distribuido permite generar una pista de auditoría fiable y resistente a la manipulación. Cada registro de sanción, incluyendo sus Metadatos (fecha, hora, ubicación, tipo de infracción) y el Hash de la evidencia asociada, puede ser anclado a la cadena, proporcionando una fuente única de verdad verificable por las partes autorizadas. Esto se alinea con los principios de Sistemas Distribuidos aplicados a la gobernanza.

Blockchains públicas vs. permissionadas. En el contexto gubernamental, la literatura y los estudios piloto tienden a favorecer las blockchains permissionadas (o de consorcio). Si bien las blockchains públicas ofrecen máxima transparencia, las permissionadas permiten a las entidades gubernamentales controlar quién puede participar en la red (validar transacciones, acceder a datos), gestionar mejor la privacidad (crucial para datos ciudadanos) y, a menudo, ofrecer mayor rendimiento y escalabilidad. La elección impacta directamente en el modelo de Confianza Descentralizada implementado.

Madurez y barreras. Aunque existen numerosos estudios y proyectos piloto (ej., registros de tierras en Suecia o Georgia, identidad digital en Estonia), las implementaciones a gran escala para la gestión integral de sanciones administrativas aún son limitadas. La madurez es variable. Las barreras reconocidas incluyen la complejidad técnica, la necesidad de marcos legales y regulatorios adaptados, la interoperabilidad con sistemas heredados, los costos iniciales de implementación y la adopción tanto por parte de las instituciones como

de los ciudadanos. La Percepción Pública de la Tecnología Blockchain también juega un rol significativo.

Integración de blockchain e IPFS para datos voluminosos y verificables

El almacenamiento directo de datos voluminosos (como imágenes o vídeos de alta resolución) en una Blockchain es ineficiente y costoso. La literatura técnica y diversos prototipos exploran la integración de Blockchain con sistemas de Almacenamiento Direccional por Contenido como IPFS (InterPlanetary File System) para abordar este desafío.

Estado actual. El enfoque predominante consiste en almacenar el dato voluminoso (la imagen del fotocomparendo) en IPFS, obteniendo un Hash único basado en su contenido. Este Hash IPFS, junto con otros Metadatos relevantes, se almacena en una Transacción Blockchain. Este modelo aprovecha la eficiencia de IPFS para el almacenamiento distribuido y la fortaleza de Blockchain para el registro inmutable y verificable del puntero (el Hash) y los metadatos asociados.

Ventajas y desafíos. Las ventajas logradas incluyen la verificabilidad (cualquier cambio en el archivo IPFS cambiaría su hash, invalidando el enlace en la Blockchain), la resiliencia potencial (si múltiples nodos almacenan el archivo) y el direccionamiento por contenido inherente a IPFS. Sin embargo, persisten desafíos persistentes significativos:

Persistencia de datos (pinning). Los datos en IPFS solo persisten mientras algún nodo los esté "pineando"(almacenando activamente). Garantizar la persistencia a largo plazo de la evidencia requiere mecanismos o servicios de pinning fiables, que pueden tener costos asociados.

Disponibilidad. La recuperación del archivo depende de que los nodos que lo almacenan estén en línea y accesibles.

Costos a largo plazo. El almacenamiento distribuido no es necesariamente gratuito, especialmente si se requieren garantías de disponibilidad y persistencia.

Gestión de la privacidad. Los datos en IPFS son típicamente accesibles públicamente si se conoce el hash. Para evidencia sensible, se requerirían capas adicionales de encriptación antes de la subida a IPFS, añadiendo complejidad.

Gestión de evidencia digital y cadena de custodia con DLT

La integridad y la cadena de custodia de la evidencia digital son cruciales en procesos sancionatorios. Blockchain/DLT ofrece mecanismos basados en Criptografía Aplicada para fortalecer estos aspectos.

Fortalecimiento de la integridad y trazabilidad. Al registrar el Hash de la evidencia digital (imagen del fotocomparendo) en una Transacción Blockchain, se crea un sello de tiempo (timestamping) inmutable y verificable. Cualquier intento posterior de modificar la evidencia original resultaría en un hash diferente, lo que permitiría detectar fácilmente la manipulación. La secuencia de transacciones en la Blockchain proporciona una trazabilidad auditable del ciclo de vida de la evidencia (captura, registro).

Comparación y valor añadido. En comparación con los sistemas tradicionales (bases de datos centralizadas, logs de servidor), que pueden ser susceptibles a alteraciones internas o fallos únicos, la DLT aporta un valor añadido significativo al distribuir la confianza y hacer que la manipulación sea computacionalmente inviable (principio de Inmutabilidad). Esto refuerza la Confianza Descentralizada en la validez de la evidencia presentada, reduciendo potenciales disputas.

Estándares emergentes. En el ámbito de la tecnología blockchain, observamos la consolidación de estándares emergentes en diversas áreas, que representan un consenso práctico y técnico en ausencia de normas formales universalmente ratificadas.

Un área clave es la Seguridad de Smart Contracts. Para construir confianza y fiabilidad en las aplicaciones descentralizadas (dApps) y mitigar vulnerabilidades, se están adoptando ampliamente prácticas que funcionan como estándares de facto:

- **Auditorías y Listas de Chequeo:** Metodologías promovidas por firmas especializadas como ConsenSys Diligence, Trail of Bits y OpenZeppelin se han vuelto

habituales.

- **Patrones de Diseño Seguro:** Se aplican convenciones como Checks-Effects-Interactions y el uso de proxies actualizables (UUPS, Transparent Proxy), aunque estos patrones continúan evolucionando.
- **Estándares de Reporte de Vulnerabilidades:** Propuestas como las EIPs (Ethereum Improvement Proposals) relacionadas con la seguridad ayudan a estandarizar la comunicación de fallos.

Otra área fundamental donde emergen estándares es la Gestión de Evidencia Digital y Cadena de Custodia mediante Blockchain/DLT. Aunque todavía no existe una norma global única (como un estándar ISO específico para esta aplicación), sí se está formando un fuerte consenso técnico sobre los principios tecnológicos clave para asegurar la integridad y fiabilidad:

- **Hashing Criptográfico:** El uso de funciones hash para generar una huella digital única e infalsificable de la evidencia (como el CID en IPFS) es la práctica estándar para garantizar la integridad y detectar manipulaciones (**benet2014ipfs**).
- **Timestamping Inmutable:** Registrar el hash de la evidencia y sus metadatos en una transacción blockchain proporciona una marca de tiempo segura e inalterable, estableciendo una prueba fehaciente del momento del registro (**nakamoto2008bitcoin**).
- **Registro en Ledger Distribuido (DLT):** Utilizar la DLT como el libro contable distribuido para estos registros es el mecanismo reconocido para lograr inmutabilidad, transparencia controlada y auditabilidad (**swan2015blockchain**), superando las limitaciones de las bases de datos centralizadas.

Implementaciones reales de blockchain en gobiernos

Registro de propiedad en Suecia

La autoridad catastral sueca *Lantmäteriet* realizó entre 2016 y 2017 un piloto con una cadena permissionada y contratos inteligentes para registrar transacciones inmobiliarias. El proyecto buscó reducir la manipulación documental y agilizar los trámites que involucran a bancos, agentes inmobiliarios y entidades estatales. Los resultados mostraron que el tiempo de compraventa podría reducirse de 4–7 meses a tan solo unos días, con un ahorro estimado de ~100 millones de euros anuales gracias a la eliminación de procesos en papel y la automatización parcial ([lantmateriet2017](#); [lantmateriet__cointelegraph2017](#); [lantmateriet__cointelegraph2__2017](#)).

Durante la segunda fase se incorporaron contratos inteligentes que ejecutaban automáticamente pasos como la firma digital de documentos y el registro de hipotecas cuando se cumplían condiciones predefinidas, demostrando la viabilidad técnica y la interoperabilidad entre actores.

Integridad de registros clínicos en Estonia

Desde 2016 la autoridad nacional de salud de Estonia, en colaboración con Guardtime, emplea la infraestructura KSI (*Keyless Signature Infrastructure*) para asegurar la integridad de los expedientes médicos de más de un millón de ciudadanos. En lugar de almacenar datos sensibles en la cadena, el sistema registra huellas *hash* de cada operación sobre la historia clínica, posibilitando auditorías en tiempo real y la detección inmediata de accesos o modificaciones no autorizadas ([guardtime2016](#); [reddit__estonia__blockchain](#)). Esta aproximación cumple los requisitos de privacidad y refuerza la confianza pública en el manejo de datos sanitarios.

Síntesis y relevancia para el proyecto

Estos casos evidencian que:

- La tecnología blockchain coordina procesos complejos con múltiples partes

interesadas, garantizando un registro único y verificable (ejemplo de Suecia).

- Permite verificar de forma irrefutable la integridad de datos sensibles sin exponer su contenido, manteniendo el cumplimiento normativo (ejemplo de Estonia).

Las lecciones aprendidas refuerzan la pertinencia de aplicar una arquitectura basada en Blockchain e IPFS para gestionar evidencias de fotocomparendos en Bogotá, buscando niveles de transparencia, inmutabilidad y confianza comparables.

Funcionamiento de los fotocomparendos en Bogotá: mecanismos, regulación e impacto

El sistema de fotocomparendos en Bogotá (FENIX) representa un modelo tecnológico y regulatorio diseñado para mejorar la seguridad vial mediante la detección automatizada de infracciones de tránsito (**mintransporte2023**). Basado en cámaras de fotodetección ubicadas en zonas autorizadas por el Ministerio de Transporte, este sistema combina vigilancia electrónica, validación humana y marcos legales específicos para sancionar conductas de riesgo (**supertransporte2021**). Desde su implementación, ha logrado reducir siniestros en puntos críticos, aunque enfrenta desafíos técnicos y jurídicos. A continuación, se detalla su operación, criterios de aplicación y marco legal que lo regula (**mintransporte2023**).

Marco legal y regulatorio. La implementación de fotocomparendos en Bogotá se sustenta en la Ley 1843 de 2017 (**ley1843**) y su reglamentación mediante resoluciones como la Resolución 20203040011245 (**resolucion11245**). Estos instrumentos establecen cuatro criterios para instalar cámaras:

1. **Siniestralidad:** Ubicación en zonas con alto índice de accidentes.
2. **Prevención:** Disuasión de conductas peligrosas.
3. **Movilidad:** Optimización del flujo vehicular.
4. **Historial de infracciones:** Enfoque en corredores con recurrentes violaciones.

Adicionalmente, las autoridades deben garantizar la visibilidad de los dispositivos, señalizando su presencia al menos 500 metros antes de su ubicación (**ley1843**), y cumplir con planes de seguridad vial alineados con políticas distritales. La Secretaría Distrital de Movilidad (SDM) ha enfrentado cuestionamientos legales, como los señalados por la Personería en 2018 (**sdm__camaras2023**).

Proceso operativo de los fotocomparendos.

Detección y captura de infracciones. Las cámaras de fotodetección en Bogotá se clasifican en dos tipos (**supertransporte2021**; **mintransporte2023**):

- **Automáticas:** Monitorean velocidades, semáforos en rojo y restricciones como pico y placa.
- **Semiautomáticas:** Vigilan bloqueos de calzadas, paradas prohibidas y recolección irregular de pasajeros.

Estos dispositivos, instalados en corredores de alta accidentalidad como la Avenida NQS o la Calle 100, capturan imágenes o videos que incluyen matrícula, fecha, hora y ubicación GPS¹⁴. Por ejemplo, en 2025, un conductor que exceda el límite de 50 km/h en la Avenida Boyacá será registrado por cámaras previamente señalizadas.

Validación y notificación. Una vez detectada una presunta infracción, las pruebas se envían a un centro de análisis de la SDM, donde agentes de tránsito verifican:

- Legibilidad de la matrícula.
- Contexto de la violación (ejemplo: si un semáforo en rojo fue respetado).
- Datos del vehículo en el RUNT (Registro Único Nacional de Tránsito).

Tras validar la infracción, se genera un comparendo electrónico notificado al propietario del vehículo mediante correo certificado o plataformas digitales. El plazo máximo para emitir la sanción es de 10 días hábiles desde la detección, seguido de 3 días para su notificación. Si

el domicilio registrado está desactualizado, el infractor podría no recibir la notificación, lo que no exime el pago (**ley1843**).

Tecnología y transparencia. El sistema combina:

- **Cámaras de última generación:** Equipadas con sensores de velocidad Lidar y visión nocturna.
- **Plataforma de análisis IA:** Algoritmos que descartan falsos positivos (ejemplo: ambulancias en emergencia).
- **Integración con RUNT:** Verificación instantánea de documentos como SOAT o tarjeta de operación.

Los datos se almacenan en servidores con cifrado AES-256, accesibles solo para funcionarios autorizados mediante autenticación biométrica.

Problemas operativos.

- **Notificaciones fallidas:** Errores en direcciones del RUNT causan sanciones no recibidas, acumulando intereses moratorios.
- **Latencia en validaciones:** En horas pico, el volumen de infracciones puede retrasar procesamiento hasta 72 horas.

Cuestionamientos legales. En 2018, la Personería de Bogotá identificó que el 15 % de las cámaras operaban sin autorización ministerial durante un periodo de transición legal. La SDM rectificó esta situación en 2019, regularizando todos los dispositivos bajo la Resolución 20203040011245 de 2020 (**secretaria__movilidad2023**).

Aplicaciones específicas en gestión de tráfico e infracciones

Al revisar las Aplicaciones de Blockchain en la Gestión de Tráfico, Infracciones y Fotocomparendos, como se observa en la Tabla , se observa que, si bien hay discusiones teóricas (**yousfi2022its**), propuestas conceptuales (**chen2024blockchain**) y hasta una

joven PYME española, las implementaciones prácticas que integren el flujo completo descrito en el prototipo (captura -> IPFS -> Blockchain -> Verificación -> Pago Automatizado con Billetera Digital) son escasas y se encuentran en fase de propuesta o son parciales (omar2024srtm; choquevilca2024blockchain).

Tabla 3. *Análisis Comparativo del Estado del Arte en Gestión de Infracciones con Blockchain*

Trabajo/Proyecto		Ámbito	Tecnologías	Limitaciones Identificadas	Aporte Relevante para el Prototipo
Yousfi et al. (2022)		Gestión de tráfico urbano	Blockchain pública, Smart Contracts	Alto costo de transacciones (gas), privacidad limitada para datos personales	Modelo conceptual de integración blockchain-tráfico, solución a la transparencia y trazabilidad
Chen et al. (2024)		Sistema de multas electrónicas	Base de datos centralizada + Blockchain	Falta de inmutabilidad completa, dependencia del servidor central	Propuesta de registrar hash de actas en blockchain para mayor integridad y transparencia

Cuadro 3 – continuación de la página anterior

Trabajo/Proyecto/Ámbito		Tecnologías	Limitaciones Identificadas	Aporte Relevante para el Prototipo
Joseph (2023)	Registros vehiculares gubernamentales	Hyperledger Fabric, IPFS	Complejidad para escalar, gestión de identidades	Arquitectura per-misionada para manejo seguro de datos sensibles
Dutta et al. (2023)	Seguros automotrices	Ethereum, Smart Contracts	Latencia en transacciones, costos operativos	Automatización de procesos mediante contratos inteligentes
Omar et al. (2024)	Gestión de infracciones de tránsito	Blockchain híbrida, base de datos	Integración parcial, falta de flujo completo	Aproximación hacia una gestión descentralizada con uso mixto de tecnologías
Choquevilca Quispe & Morales Valencia (2024)	Fotocomparendo en Latinoamérica	Análisis conceptual	Falta de implementaciones prácticas en la región	Identificación de brechas y oportunidades para implementación blockchain

Cuadro 3 – continuación de la página anterior

Trabajo/Proyecto	Ámbito	Tecnologías	Limitaciones Identificadas	Aporte Relevante para el Prototipo
Proyectos e-gov (Suecia, Estonia)	Registros gubernamentales	Blockchain permisionada, KSI	Limitado a registros específicos, no multas	Validación técnica y mejora en eficiencia para registros oficiales
Anand & Singh (2024)	Gestión de documentos oficiales	IPFS + Blockchain	Persistencia en IPFS, costos de almacenamiento	Almacenamiento distribuido para evidencias con verificación en blockchain
JUIT Research Group (2024)	Sistema de pagos gubernamentales	Stablecoins, Smart Contracts	Adopción de criptomonedas, entorno regulatorio restringido	Automatización de pagos en ecosistemas blockchain

Nota. Elaboración propia.

Análisis de existencia. La literatura existente se centra más en componentes aislados (yousfi2022its), uso de blockchain para registros vehiculares (mani2023smart), seguros (dutta2023solution), o gestión genérica de multas (omar2024srtm), pero raramente combinando el almacenamiento de evidencia en IPFS con la automatización del pago vía billetera digital específicamente para fotocomparendos.

Arquitecturas y resultados. Dada la escasez de implementaciones completas reportadas ([AnandSingh_ProjectReport_Year](#); [juit2024traffic](#)), es difícil generalizar sobre arquitecturas dominantes o resultados concretos para este caso de uso tan específico. Los estudios existentes a menudo se limitan a explorar la viabilidad teórica o a implementar módulos parciales ([choquevilca2024blockchain](#)).

Lecciones aprendidas. La principal lección aprendida de áreas adyacentes es la importancia de abordar no solo los desafíos técnicos (Zheng et al., 2018) sino también los regulatorios, de gobernanza y de adopción (Tan et al., 2022). La ausencia de soluciones integrales reportadas para el flujo completo de gestión de fotocomparendos representa una brecha significativa en la aplicación práctica de estas tecnologías combinadas.

El análisis del estado del arte revela avances significativos, pero también limitaciones claras:

Avances significativos

La tecnología Blockchain ha demostrado su potencial para crear registros gubernamentales más inmutables, transparentes y auditables. (Balcerzak et al., 2022; Meroni et al., 2023)

La integración Blockchain + IPFS es una solución técnicamente viable y reconocida para gestionar datos voluminosos referenciados desde una cadena de bloques, mejorando la verificabilidad (Adel et al., 2023; Mishra et al., 2024).

DLT ofrece mejoras sustanciales para la integridad y trazabilidad de la evidencia digital (Thanasas et al., 2025).

Existen mecanismos (billeteras digitales, smart contracts, stablecoins) para habilitar pagos digitales automatizados en ecosistemas blockchain ([antonopoulos2023mastering](#)).

Limitaciones identificadas

Madurez e integración. Muchas aplicaciones gubernamentales de Blockchain son pilotos aislados (Zheng et al., 2018; Li et al., 2021). Falta integración entre sistemas y con procesos completos.

Desafíos técnicos. La persistencia y gestión a largo plazo de datos en IPFS (pinning), la escalabilidad de algunas blockchains y la seguridad/fiabilidad de los oráculos para Smart

Contracts siguen siendo áreas de desarrollo activo (Zheng et al., 2018).

Adopción y regulación. La adopción de billeteras digitales para pagos gubernamentales, el uso de criptoactivos/stablecoins y la claridad legal sobre smart contracts en el sector público son obstáculos importantes (Tan et al., 2022).

Política de reserva de información. Bogotá mantiene una política de reserva de información que restringe la divulgación completa de los datos almacenados en su base de datos. Esta política limita el acceso y la difusión de ciertos datos, lo que puede afectar la transparencia y la capacidad de realizar un análisis exhaustivo (choquevilca2024blockchain).

Actualización de datos. La actualización de los datos almacenados en la base de datos es un proceso que requiere tiempo. Dada la naturaleza progresiva de este proceso, que depende de la cantidad de datos que se agregan diariamente, la actualización completa de los datos con el sistema que se desarrollará puede llevar un tiempo considerable (choquevilca2024blockchain).

Aplicación específica. Existe una notable ausencia de soluciones documentadas que implementen el flujo completo e integrado (captura de imagen -> subida a IPFS -> registro en Blockchain -> verificación vía app -> pago automático desde billetera digital) específicamente para la gestión de fotocomparendos. (yousfi2022its; chen2024blockchain)

Verificación participativa. Los sistemas actuales raramente permiten que el ciudadano verifique independientemente la autenticidad e integridad de la evidencia presentada contra ellos, limitando los beneficios de transparencia inherentes a blockchain.

Adopción en Bogotá. La literatura muestra una escasez notable de implementaciones o estudios piloto en contextos latinoamericanos, donde factores como confianza institucional, infraestructura tecnológica y marcos regulatorios presentan desafíos particulares. (choquevilca2024blockchain; rezabala2025blockchain)

Novedad y relevancia del prototipo

La(s) brecha(s) específica(s) que este prototipo busca abordar es precisamente la falta de una solución integrada y de extremo a extremo para la gestión de fotocomparendos utilizando la sinergia de Blockchain, IPFS y pagos automatizados, como se evidencia en la revisión de la literatura existente ([yousfi2022its](#); [AnandSingh_ProjectReport_Year](#))

La novedad principal radica en la integración holística de todo el flujo propuesto. Mientras que los componentes individuales han sido explorados por separado (Adel et al., 2023; Mishra et al., 2024) o en otros contextos (Mani Joseph P, 2023; Dutta et al., 2023), este prototipo propone conectarlos en una secuencia lógica y automatizada para este caso de uso particular.

Aborda la brecha de aplicación específica, llevando los conceptos teóricos ([swan2015blockchain](#); [antonopoulos2023mastering](#)) y las soluciones parciales existentes ([choquevilca2024blockchain](#)) a un dominio concreto (fotocomparendos en Bogotá) con un proceso completo.

La relevancia del prototipo se justifica por su potencial para:

Mejorar la transparencia y confianza en el proceso de fotocomparendos, evidencia verificable e inmutable (Meroni et al., 2023; Thanasas et al., 2025).

Aumentar la eficiencia operativa mediante la automatización del registro, verificación y pago.

Reducir disputas y costos asociados a la gestión manual y a la falta de confianza en la evidencia.

Explorar un modelo innovador de pago automatizado condicional basado en la verificación en Blockchain.

Análisis de tendencia internacional

Producción científica por países (mapa y gráfico de líneas). Descripción

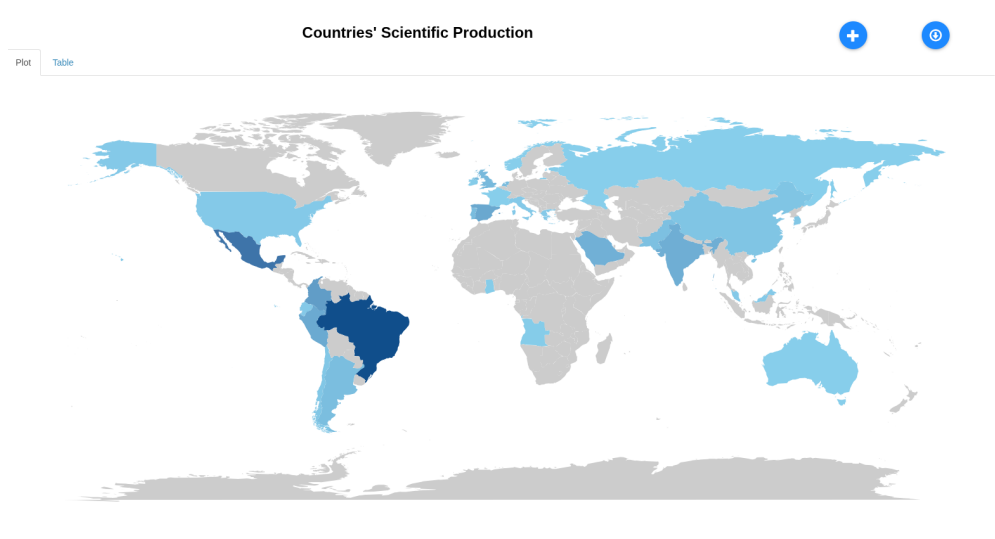
General: Esta gráfica se compone de dos partes. La primera es un mapa mundial que utiliza una escala de color para representar la cantidad de producción científica por país.

Las tonalidades más oscuras generalmente indican una mayor producción. La segunda parte es un gráfico de líneas que muestra la evolución de la producción científica (en artículos) a lo largo de los años para un conjunto específico de países.

Mapa mundial. El mapa muestra la distribución global de la producción científica en el área de estudio. Se observa una concentración significativa de publicaciones en países como Brasil, lo que sugiere un interés y actividad investigadora importante en Latinoamérica. Otros países con una producción notable incluyen México y España. Es importante notar que algunas regiones muestran una menor actividad, lo que podría indicar diferencias en el enfoque de investigación, financiamiento o acceso a recursos.

Figura 2

Distribución global de la producción científica sobre blockchain y gestión de infracciones

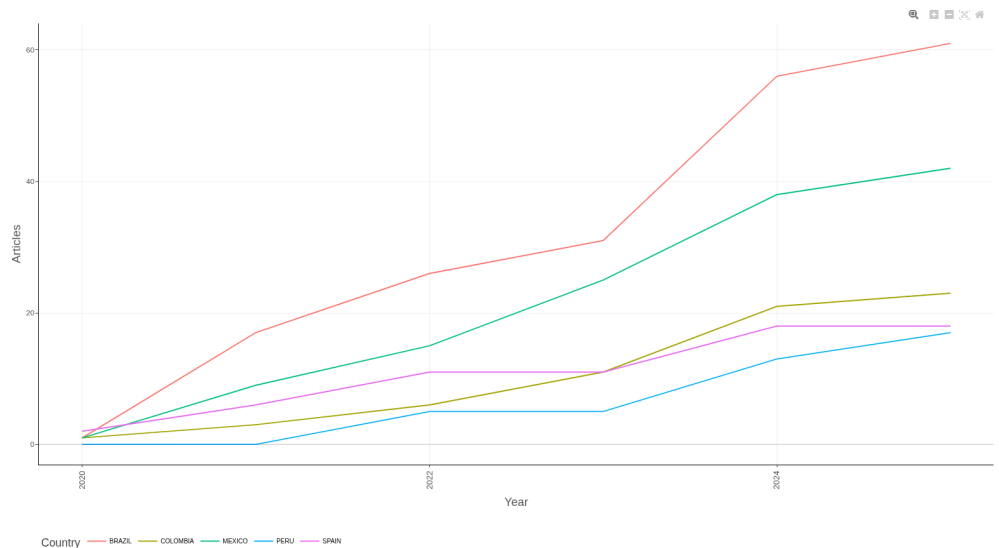


Nota. Elaboración propia con datos bibliométricos.

Nube de palabras. La Figura 4 muestra los términos más frecuentes en la literatura analizada. Destacan conceptos como *blockchain*, *challenges*, *management* y *secure*, lo que

Figura 3

Evolución anual de publicaciones en los países líderes del tema (Brasil, México, Colombia, España y Perú)



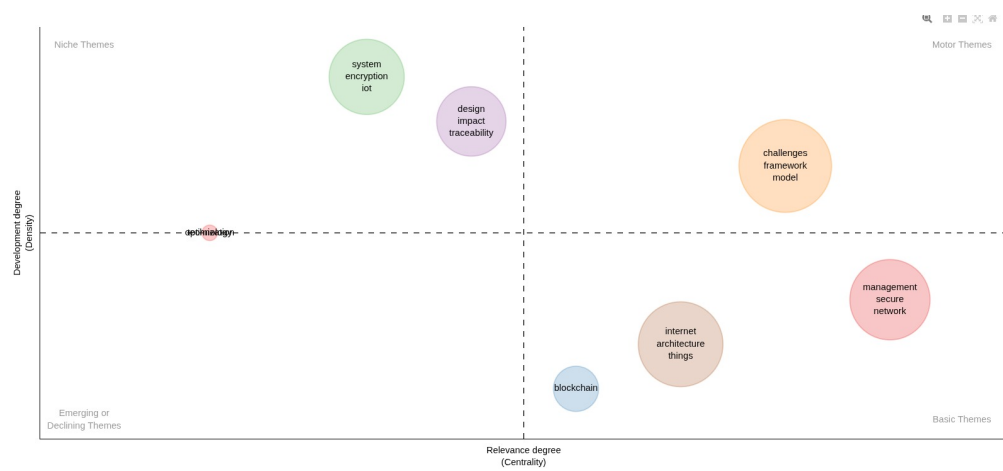
Nota. Elaboración propia con datos bibliométricos.

refleja el énfasis de la comunidad académica en los retos de seguridad y gestión al aplicar tecnologías DLT en contextos gubernamentales.

Mapa temático. El mapa temático representa la distribución de los principales temas de investigación en el área, organizados según su grado de desarrollo (densidad) y relevancia (centralidad). En el cuadrante superior derecho se ubican los "temas motores", como "challenges, framework, model", que son altamente desarrollados y centrales en la literatura. En el cuadrante inferior derecho, los "temas básicos" como "management, secure, networkz internet, architecture, things" son fundamentales pero menos desarrollados. El cuadrante superior izquierdo agrupa "temas nicho" como "system, encryption, iotz "design, impact, traceability", que presentan alta especialización pero menor centralidad.

Figura 5

Mapa temático de los principales temas de investigación en el área



Nota. Elaboración propia con datos bibliométricos.

Alcance

Enfoque y delimitación geográfica

Este trabajo se circunscribe al proceso de generación, gestión y verificación de **multas de tránsito automatizadas (fotomultas)** emitidas por la Secretaría Distrital de Movilidad de Bogotá. Se excluyen deliberadamente:

- Multas impuestas de forma presencial por agentes de tránsito.
- Procesos sancionatorios de otras ciudades o entidades territoriales.
- Funcionalidades de recaudo y pasarelas de pago (solo se registra el estado del pago, no se procesa el pago en sí).

Componentes del prototipo

El prototipo aborda los siguientes módulos funcionales:

1. **Registro inmutable de la infracción** Captura de metadatos (placa, fecha, hora, ubicación y tipo de infracción) y publicación del identificador de la evidencia en la *blockchain* (Ethereum local con Hardhat para desarrollo, con arquitectura preparada para Hyperledger Fabric en producción).
2. **Almacenamiento descentralizado de evidencias** Carga de la imagen o video de la fotomulta en IPFS y obtención de su *hash*.
3. **Verificación pública** Servicio de consulta que permite contrastar el hash guardado en la cadena con el archivo almacenado en IPFS.
4. **Gestión del ciclo de vida de la multa** Estados: Generada → Notificada → En apelación → Pagada → Cerrada. Cada transición queda registrada mediante eventos de contrato inteligente.
5. **Interfaz mínima** Panel Web para: (i) agentes que registran la infracción y (ii) ciudadanos que consultan la autenticidad y el estado de su fotomulta.

Fuera del alcance

- Integración completa con sistemas legados del RUNT o SIMIT; se simula mediante datos de prueba.
- Implementación de un modelo económico (tarifas de gas, costos operativos reales).
- Implementación de algoritmos de detección automática de infracciones (visión por computador). Se parte de que la cámara ya detectó la infracción y generó la evidencia.

Entregables

- Contrato inteligente en Solidity con 80 pruebas automatizadas (97.5 % de éxito).
- Script de despliegue de red Ethereum local (Hardhat) e instalación de IPFS local.
- Aplicación Web de demostración (*frontend* ligero) conectada a los servicios anteriores.
- Manual técnico que documenta la arquitectura y el flujo de datos.
- Informe de resultados de las pruebas funcionales y de rendimiento básico.

Criterios de éxito

1. Tiempo medio de publicación de una infracción ≤ 3 s en entorno de laboratorio.
2. Coincidencia 100 % entre el hash almacenado en la cadena y la evidencia recuperada desde IPFS.
3. Trazabilidad completa del historial de estados para al menos 50 multas de prueba.
4. Ausencia de fallos críticos en pruebas de carga con 10 transacciones concurrentes.

Limitaciones del prototipo

Es fundamental reconocer que, como prototipo desarrollado en un contexto académico, el presente estudio presenta ciertas limitaciones que definen el alcance de sus conclusiones y delinean claras oportunidades para futuras investigaciones. Las principales limitaciones son:

1. Entorno de validación

- **Validación en Entorno de Laboratorio:** El prototipo fue diseñado, desplegado y evaluado en un entorno de simulación controlado. No se sometió a pruebas en una infraestructura productiva real con la carga de transacciones y el volumen de usuarios que gestiona actualmente la Secretaría de Movilidad. Por lo tanto, su rendimiento, estabilidad y escalabilidad bajo condiciones de estrés real aún no han sido cuantificados.
- **Uso de Datos Simulados:** Debido a estrictas normativas de privacidad y protección de datos personales que impiden el acceso a información real de ciudadanos y vehículos, todas las pruebas se realizaron con datos sintéticos. Esto implica que el prototipo no fue expuesto a la variabilidad, inconsistencias y casos atípicos que caracterizan a los datos del mundo real, lo cual podría influir en la lógica de negocio y en el manejo de errores en un entorno de producción.
- **Suposiciones sobre la Calidad de la Evidencia:** El sistema asume que las evidencias fotográficas (imágenes de fotocomparendos) son capturadas con una calidad suficiente para su procesamiento. No se implementaron ni probaron mecanismos para manejar escenarios con imágenes de baja resolución, borrosas o con obstrucciones, que son comunes en la operación real.

2. Integración y comparación con sistemas existentes

- **Integración Simulada con Sistemas Externos:** La interacción con plataformas gubernamentales clave como el RUNT y el SIMIT fue simulada a través de APIs de prueba (mocks). No se abordaron los desafíos técnicos y burocráticos de una integración real, como los protocolos de comunicación, los tiempos de respuesta, la disponibilidad de los servicios y los posibles cuellos de botella.
- **Ausencia de Benchmarking Directo con el Sistema Actual (Fénix):** La

falta de acceso al código fuente y a la arquitectura interna del sistema Fénix impidió realizar una comparación cuantitativa y directa en términos de rendimiento, costos operativos o eficiencia de procesos. El análisis comparativo se basó en las características conceptuales de ambas arquitecturas (centralizada vs. descentralizada).

3. Aspectos técnicos y de escalabilidad

- **Proyección de Costos como Escenario de Referencia:** Los costos de infraestructura y desarrollo estimados corresponden a un escenario de referencia. Los costos reales en un despliegue a gran escala podrían variar considerablemente dependiendo de factores como el número de nodos en la red, el volumen de almacenamiento en IPFS, el tráfico de red y la estrategia de persistencia de datos (pinning) que se adopte.
- **Estrategia de Persistencia en IPFS:** Para que la evidencia digital permanezca disponible a largo plazo en IPFS, es necesario que al menos un nodo la mantenga “pineada”. El prototipo no implementa una política de pinning distribuida y resiliente, lo cual sería un requisito crítico para garantizar la cadena de custodia digital en un sistema de producción.

4. Seguridad y robustez

- **Limitaciones en Pruebas de Seguridad Avanzadas:** Si bien el prototipo implementa validaciones básicas de entrada (XSS, SQL injection, path traversal) y manejo de errores a nivel de aplicación, validadas mediante 26 pruebas automatizadas con 100 % de éxito, el alcance del proyecto **no contempló auditorías de seguridad exhaustivas** como las siguientes:
 - **Análisis estático de contratos inteligentes:** No se emplearon herramientas especializadas como *Slither*, *Mythril* o *MythX* para detectar vulnerabilidades en el código Solidity del contrato **FineRegistry**.

- **Pruebas de penetración (pentesting):** No se realizaron ataques simulados avanzados sobre la API REST más allá de las validaciones básicas implementadas.
- **Auditoría de permisos y autenticación:** El prototipo actual implementa validaciones básicas de entrada pero no incluye un sistema robusto de autenticación y autorización (JWT, OAuth2, RBAC) necesario para producción.
- **Validación exhaustiva de límites de archivos IPFS:** Aunque se validan formatos de imagen (JPG, PNG, WEBP) y límites de tamaño (10MB), no se implementaron validaciones avanzadas contra contenido malicioso embebido (steganografía, malware).
- **Protección contra ataques de denegación de servicio (DoS):** No se implementaron mecanismos de *rate limiting*, *throttling* o *CAPTCHA* para prevenir abusos en los endpoints públicos.

Trabajo Futuro en Seguridad. Se recomienda que, en fases posteriores de desarrollo hacia producción, se incorporen las siguientes medidas:

- a) Integración de herramientas de análisis estático como *Slither* para contratos Solidity.
- b) Implementación de un sistema de autenticación y autorización basado en roles (RBAC) con tokens JWT.
- c) Auditoría de seguridad externa realizada por especialistas en *blockchain security*.
- d) Pruebas de penetración automatizadas utilizando herramientas como *OWASP ZAP* o *Burp Suite*.
- e) Implementación de validación de archivos mediante *magic numbers* y análisis de contenido.

f) Configuración de límites de tasa (*rate limiting*) en la API REST.

Estas limitaciones **no comprometen la validez de la prueba de concepto**, dado que el objetivo principal es demostrar la viabilidad técnica de un modelo de confianza descentralizado basado en inmutabilidad y verificabilidad, no el despliegue de un sistema en producción listo para operar en un entorno real con amenazas activas.

Metodología

La metodología de este proyecto se divide en dos componentes principales: la metodología de investigación y la metodología de desarrollo de software.

Metodología de investigación

La investigación se clasifica de la siguiente manera:

- En función de su aplicación práctica, corresponde a una investigación aplicada, pues se orienta a resolver la falta de seguridad que existe en la gestión de infracciones de tránsito en la ciudad de Bogotá. De acuerdo con (coulouris2011), la investigación aplicada o técnica se centra en ofrecer soluciones concretas o generar innovaciones y mejoras en procesos o productos.
- Por su propósito, el estudio es de tipo descriptivo, ya que pretende identificar y detallar las características más relevantes de la implementación de un servicio web basado en Blockchain, con el objetivo de reforzar la seguridad en la gestión de las infracciones de tránsito en dicha municipalidad. Como señala (vanSteen2017), los estudios descriptivos buscan especificar las propiedades, características y aspectos significativos del fenómeno que se analiza.

Metodología de desarrollo de software: enfoque por prototipos

Para el desarrollo de este proyecto, se adoptará la Metodología de Desarrollo por Prototipos. Esta elección se fundamenta en la naturaleza innovadora del proyecto, que combina tecnologías emergentes como Blockchain e IPFS en un dominio específico (gestión de fotocomparendos), donde los requisitos exactos y los desafíos técnicos pueden no ser completamente evidentes desde el inicio. La metodología por prototipos es inherentemente iterativa y se centra en la construcción rápida de versiones funcionales (prototipos) del sistema, permitiendo la validación temprana de conceptos, la recopilación de retroalimentación continua y la adaptación flexible a los descubrimientos realizados durante el desarrollo.

Introducción a los artefactos técnicos del diseño

Con el fin de estructurar de manera clara el desarrollo de la solución propuesta, en esta sección se presentan los principales artefactos utilizados durante la etapa de diseño. Estos elementos permiten representar gráficamente tanto la lógica de funcionamiento como la arquitectura del sistema, sirviendo como guía para la implementación y posterior validación del prototipo.

El conjunto de diagramas que se incluye responde a la necesidad de modelar distintos aspectos del sistema. Por un lado, se usan diagramas de casos de uso para identificar las funcionalidades clave desde la perspectiva del usuario. Por otro, los diagramas de clases permiten definir la estructura del software, mientras que los diagramas de despliegue muestran cómo se distribuyen los componentes en el entorno tecnológico. Además, se incluyen diagramas de flujo que describen el comportamiento del sistema ante eventos específicos, facilitando la comprensión de su dinámica interna.

Cada uno de estos artefactos está alineado con los objetivos del proyecto y fue elaborado considerando tanto las necesidades funcionales como las características propias de las tecnologías involucradas, en particular el uso de Blockchain e IPFS. De esta forma, se busca garantizar coherencia técnica en el diseño y establecer una base sólida para el desarrollo e implementación de la solución.

Diseño del prototipo

Se hace mención de que, aunque la documentación para elaborar el software está en español, es un estándar escribir código en inglés y, por tanto, para mantener la coherencia, los diagramas mostrados a continuación usarán este idioma para los nombres de las variables, funciones y clases.

Definición de requisitos

1. **Datos sobre infracciones de tráfico:** La captura de datos detallados sobre infracciones de tráfico, como la hora de la infracción, las coordenadas GPS, el tipo de infracción, los datos de identificación del vehículo e imágenes o vídeos, garantiza que cada incidente se documenta exhaustivamente. Este registro exhaustivo proporciona transparencia y responsabilidad, ya que los datos son inmutables y a prueba de manipulaciones una vez almacenados en la cadena de bloques. La inclusión de pruebas mediáticas refuerza aún más la credibilidad y verificabilidad de cada infracción, haciendo que los registros sean sólidos a efectos legales y administrativos.
2. **Información sobre el conductor:** Asociar las infracciones de tráfico a conductores concretos utilizando su dirección Ethereum (clave pública), los datos KYC si es necesario, y los números de identificación del conductor permite un seguimiento y una rendición de cuentas precisos. Esta vinculación permite al sistema personalizar el seguimiento y la verificación de las sanciones, garantizando que las sanciones se atribuyan correctamente a las personas adecuadas. El uso de datos KYC garantiza que las identidades de los conductores puedan verificarse de forma fiable, lo que resulta esencial para mantener la integridad y fiabilidad del sistema.
3. **Datos de la sanción:** Registrando los datos de la sanción, incluyendo el tipo de sanción, el importe de la sanción y el estado del pago de la sanción facilita la ejecución automatizada de las sanciones a través de contratos inteligentes. Esta automatización reduce la carga administrativa de y garantiza que las sanciones se

apliquen de forma coherente y transparente. El registro inmutable de las sanciones y su estado de pago en la blockchain garantiza que el proceso sea justo y responsable, proporcionando una pista de auditoría clara para todas las transacciones financieras relacionadas con las infracciones de tráfico.

4. **Eventos de contratos inteligentes:** El registro de eventos de contratos inteligentes, como el registro de nuevas infracciones de tráfico o la ejecución de sanciones, con datos relevantes y marcas de tiempo, garantiza que todas las acciones significativas se documenten de forma transparente. Este registro de eventos mejora la trazabilidad y la rendición de cuentas, proporcionando un registro cronológico de las actividades importantes del sistema. Esta transparencia es crucial para las auditorías y revisiones, ya que ayuda a generar confianza en las operaciones del sistema.
5. **Datos de las transacciones de la cadena de bloques:** El seguimiento de los datos de las transacciones de la cadena de bloques, incluido el hash de la transacción, las direcciones del remitente/receptor y las tarifas del gas, proporciona un registro detallado de todas las interacciones dentro del sistema. Estos datos permiten supervisar y auditar las transacciones, garantizando la transparencia y la trazabilidad. Además, hacer un seguimiento de las tarifas de gas ayuda a gestionar y optimizar los costes asociados a la ejecución de transacciones en la blockchain, que es importante para mantener la rentabilidad del sistema.
6. **Dispositivos de datos IoT:** La integración de datos de dispositivos IoT, como sensores o cámaras, junto con marcas de tiempo e identificación del dispositivo, puede mejorar las pruebas recopiladas para infracciones de tráfico. Estos datos en tiempo real proporcionan contexto adicional y pruebas corroborativas, haciendo que los registros de infracciones sean más sólidos y fiables. El uso de dispositivos IoT también puede automatizar la detección y el registro de infracciones, aumentando la eficiencia y la precisión del sistema.

7. **Opiniones de los usuarios:** La recopilación de opiniones de los usuarios, incluidos el tipo de opinión, los comentarios y las valoraciones de los usuarios, ayuda a los administradores del sistema a comprender las experiencias y percepciones de los usuarios. Esta información es valiosa para identificar áreas de mejora en y mejorar la usabilidad y funcionalidad del sistema. Involucrar a los usuarios de esta manera puede conducir a un diseño del sistema más centrado en el usuario, mejorando la satisfacción y la eficacia general.
8. **Datos de cumplimiento:** El registro de los datos de cumplimiento, incluido el estado de cumplimiento y los detalles normativos, garantiza que el sistema se adhiere a las leyes y normativas de tráfico locales. Este seguimiento es vital para demostrar el cumplimiento de la normativa y evitar problemas legales. El mantenimiento de registros de cumplimiento detallados también facilita las auditorías reglamentarias en, proporcionando pruebas transparentes de que el sistema funciona dentro de las normas legales, lo que es esencial para generar confianza y credibilidad entre las partes interesadas.

Diagrama de casos de uso del sistema de gestión de infracciones de tránsito

Diagrama de despliegue

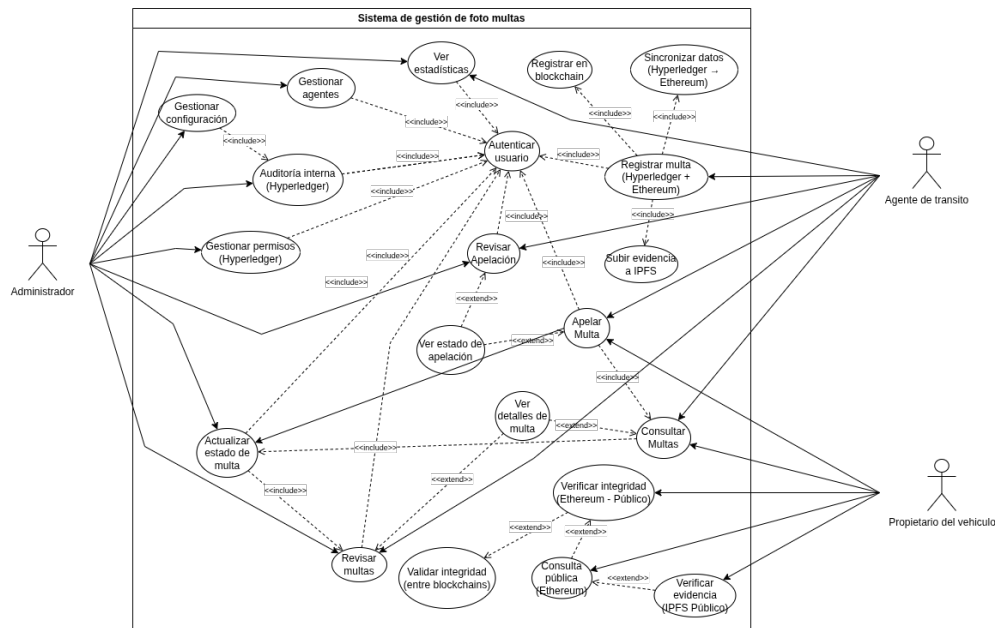
En la Figura 7 se presenta la arquitectura híbrida propuesta, que combina una red blockchain privada (Hyperledger Fabric) con una blockchain pública (Ethereum), implementando el concepto de transparencia selectiva. La arquitectura se compone de los siguientes elementos:

Capa privada - Hyperledger Fabric.

- **Nodos Peer:** Mantienen el ledger privado y ejecutan chaincode (lógica de negocio). Estos nodos almacenan la información completa de las infracciones, incluyendo datos sensibles y evidencias completas.

Figura 6

Diagrama de casos de uso del sistema de gestión de infracciones de tránsito



Nota. Elaboración propia.

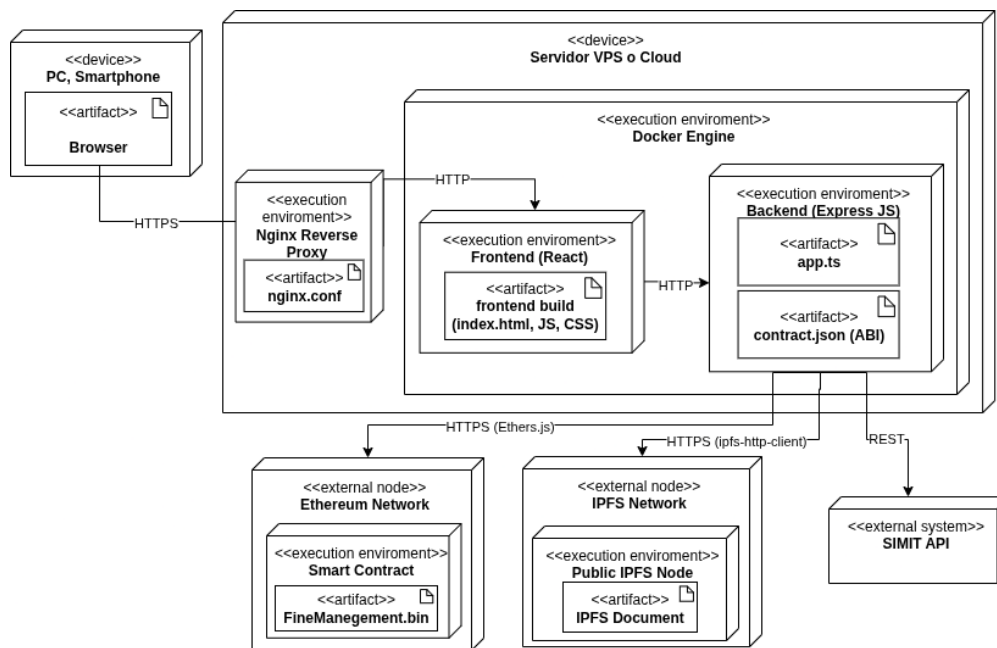
- **Nodo Orderer:** Coordina el consenso entre peers utilizando el algoritmo PBFT (Practical Byzantine Fault Tolerance), garantizando la validación eficiente de transacciones.
- **Certificate Authority (CA):** Gestiona las identidades digitales y permisos de usuarios autorizados (administradores, agentes de tránsito).
- **IPFS Privado:** Almacena las evidencias fotográficas completas con sus metadatos, accesible solo para usuarios autorizados.

Capa pública - Ethereum.

- **Nodos Ethereum:** Ejecutan Smart Contracts que almacenan metadatos públicos de

Figura 7

Diagrama de despliegue de la arquitectura del sistema



Nota. Elaboración propia.

infracciones sin información personal sensible.

- **IPFS Público:** Almacena hashes de evidencias para verificación ciudadana, sin exponer imágenes completas.

Servicio de sincronización. Un servicio intermediario sincroniza los datos entre ambas blockchains, extrayendo metadatos no sensibles de Hyperledger Fabric y publicándolos en Ethereum junto con hashes de integridad. Este servicio garantiza la consistencia entre ambas capas mediante verificación cruzada de hashes criptográficos.

La arquitectura se conecta mediante servicios web a APIs externas como Aplitude para acceder a información del Registro Único Nacional de Tránsito (RUNT) y del Sistema

Integrado de Información sobre Multas y Sanciones por Infracciones de Tránsito (SIMIT), obteniendo datos de conductores, vehículos y el estado de multas. Esta integración permite validar la información de infracciones contra registros oficiales sin comprometer la privacidad de los datos almacenados en la capa privada.

Diagrama de clases

La arquitectura de clases del sistema implementa el patrón Controller-Service-Repository, adaptado para soportar la arquitectura híbrida blockchain. Se distinguen tres capas principales de lógica de negocio:

Primera capa - servicios de blockchain híbrida. Gestiona la interacción con ambas blockchains de forma independiente:

- **HyperledgerService:** Coordina operaciones con la red privada de Hyperledger Fabric, incluyendo registro completo de infracciones, gestión de apelaciones y control de acceso.
- **EthereumService:** Maneja la publicación de metadatos en la blockchain pública de Ethereum y proporciona interfaces de consulta ciudadana.
- **SyncService:** Implementa la lógica de sincronización entre blockchains, extrayendo metadatos públicos de Hyperledger y publicándolos en Ethereum con hashes de integridad.

Segunda capa - almacenamiento distribuido dual. Separa el almacenamiento de evidencias según su nivel de sensibilidad:

- **IPFSPrivateService:** Gestiona el almacenamiento de evidencias completas en IPFS privado, accesible solo para usuarios autorizados.
- **IPFSPublicService:** Maneja la publicación de hashes de evidencias en IPFS público para verificación ciudadana.

Tercera capa - orquestación y administración. Coordina las operaciones entre todas las capas:

- **FineService:** Orquesta el flujo completo de registro de infracciones, coordinando el almacenamiento en IPFS privado, registro en Hyperledger Fabric y sincronización a Ethereum.
- **FineController:** Expone endpoints REST para las operaciones del sistema, diferenciando entre operaciones internas (requieren autenticación) y consultas públicas.

En la Figura 8 se presenta el diagrama de clases completo del sistema, mostrando la separación entre servicios de Hyperledger Fabric (privado), Ethereum (público) y el servicio de sincronización que coordina la interoperabilidad entre ambas blockchains. El diagrama ilustra cómo el patrón Repository abstrae el acceso a cada blockchain mediante repositorios especializados (HyperledgerRepository y EthereumRepository), mientras que el patrón Service encapsula la lógica de negocio específica de cada capa.

En la Figura 9 se presenta el flujo de proceso para la gestión de apelaciones de multas en la arquitectura híbrida. Este proceso se ejecuta en la capa privada de Hyperledger Fabric, garantizando la confidencialidad de las evidencias presentadas por el ciudadano. Una vez resuelta la apelación por el agente autorizado, el cambio de estado se sincroniza a la blockchain pública de Ethereum, permitiendo que el ciudadano verifique la resolución sin acceder a información sensible del proceso interno.

Diagrama de actividades

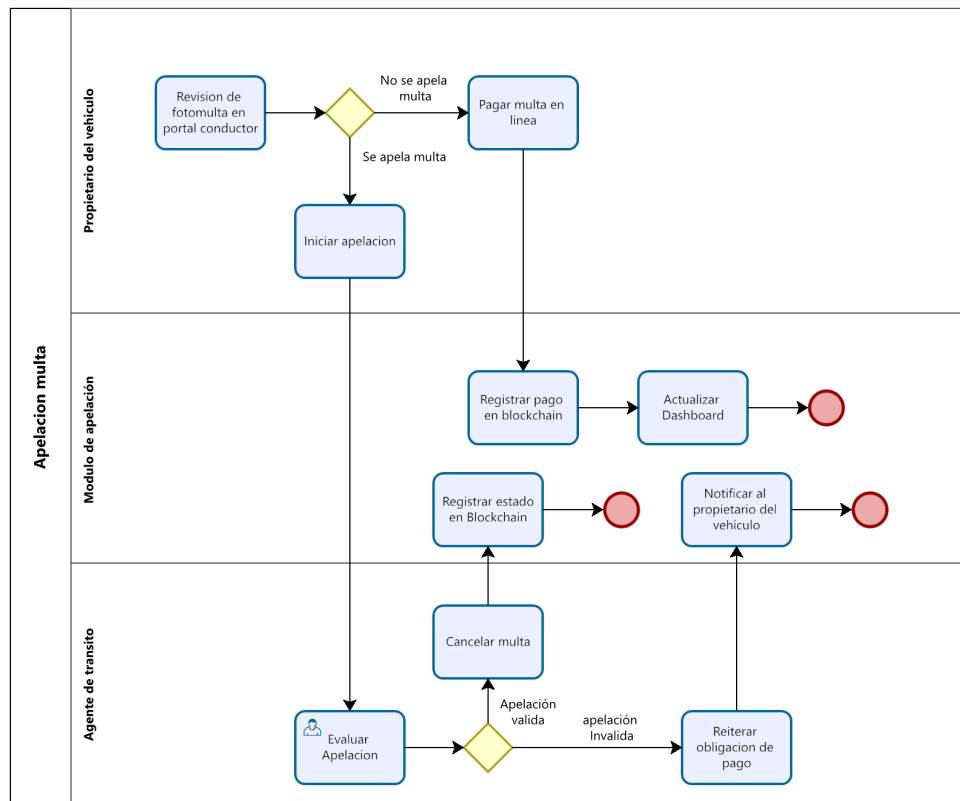
Interfaz de usuario

Compartidas. En la Figura 11 se aprecia la pantalla de inicio de sesión, punto de entrada para todos los usuarios autorizados del sistema.

La Figura 12 muestra el formulario para recuperar la contraseña, reforzando la experiencia de autoservicio y seguridad de la plataforma.

Figura 9

Diagrama de actividades para el proceso de apelación de multa

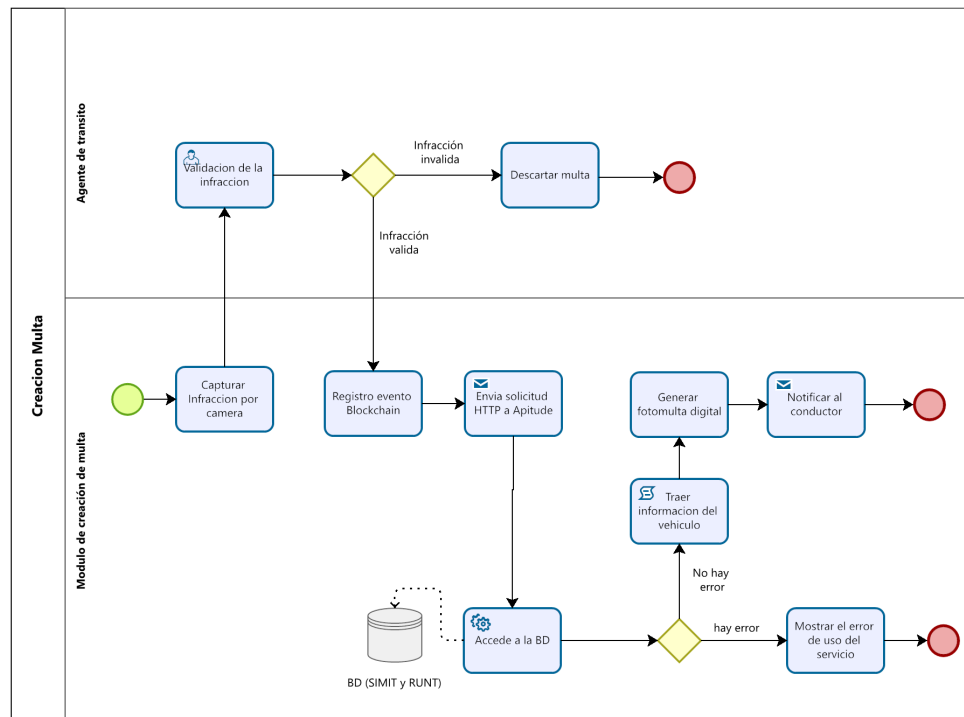


Powered by
b3xag3
Modeler

Nota. Elaboración propia.

Figura 10

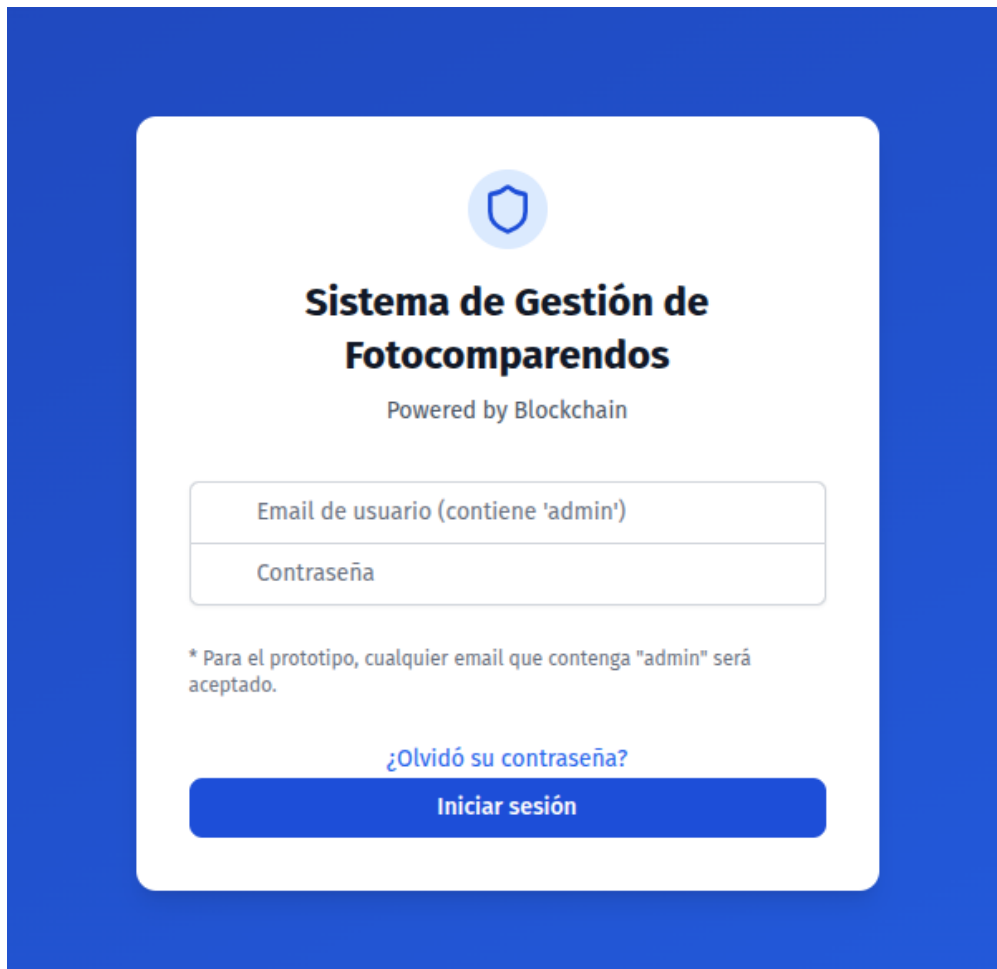
Diagrama de actividades para el proceso de creación de multa




Nota. Elaboración propia.

Figura 11

Pantalla de login del sistema





**Sistema de Gestión de
Fotocomparendos**

Powered by Blockchain

Email de usuario (contiene 'admin')

Contraseña

* Para el prototipo, cualquier email que contenga "admin" será aceptado.

[¿Olvidó su contraseña?](#)

Iniciar sesión

Nota. Elaboración propia.

Figura 12

Pantalla de recuperación de contraseña

La imagen muestra una interfaz de usuario para la recuperación de contraseña. El título principal es "Recuperar contraseña" en un color azul oscuro. Debajo del título, hay un texto de instrucción: "Ingrese su correo electrónico y le enviaremos las instrucciones". A continuación, hay un campo de entrada de texto con el label "Correo electrónico" y el texto de ejemplo "usuario@ejemplo.com". Debajo del campo de entrada, hay un botón azul con el texto "Enviar instrucciones" y un ícono de correo electrónico. En la parte inferior, hay un enlace azul que dice "Volver al inicio de sesión".

Recuperar contraseña

Ingrese su correo electrónico y le enviaremos las instrucciones

Correo electrónico

usuario@ejemplo.com

 **Enviar instrucciones**

[Volver al inicio de sesión](#)

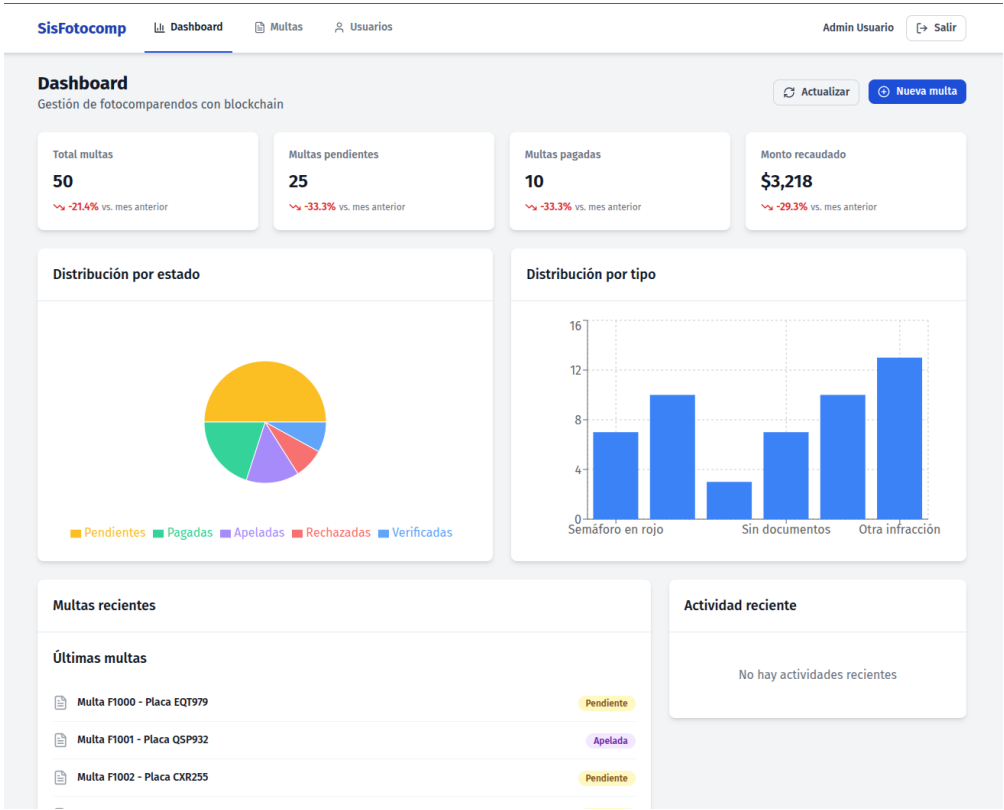
Nota. Elaboración propia.

Vista agente. En la Figura 13 se presenta el tablero principal que resume las métricas de gestión de multas para el agente de tránsito. La Figura 14 ilustra la consulta rápida del estado de una multa, facilitando el seguimiento por parte del agente. En la Figura 15 se muestra el detalle completo de una multa específica, incluida la evidencia asociada.

Vista propietario de vehículo. Por último, la Figura 16 exhibe la vista que permite al propietario del vehículo revisar todas sus multas pendientes o en proceso.

Figura 13

Dashboard del agente de tránsito



Nota. Elaboración propia.

Figura 14

Pantalla de consulta del estado de multa

SisFotocomDashboardMultasUsuariosAdmin UsuarioSalir

Gestión de multasAdministra todas las infracciones registradas en el sistemaRegistrar nueva multa

Buscar por ID o placa...

Filtros

ID	PLACA	FECHA	TIPO	MONTO	ESTADO	
F1041	KMJ917	11 de jun de 2025, 08:27 a. m.	Estacionamiento ilegal	\$ 132.279	Pendiente	Ver detalles
F1034	ABS169	11 de jun de 2025, 08:27 a. m.	Sin documentos	\$ 170.850	Pendiente	Ver detalles
F1036	SGS466	11 de jun de 2025, 08:27 a. m.	Exceso de velocidad	\$ 172.674	Cancelada	Ver detalles
F1005	VBC893	10 de jun de 2025, 08:27 a. m.	Exceso de velocidad	\$ 210.525	Pendiente	Ver detalles
F1027	ZNN678	9 de jun de 2025, 08:27 a. m.	Conducción bajo influencia	\$ 226.487	Pagada	Ver detalles
F1006	WGD43	9 de jun de 2025, 08:27 a. m.	Sin documentos	\$ 570.843	Apelada	Ver detalles
F1028	ZFZ70	8 de jun de 2025, 08:27 a. m.	Exceso de velocidad	\$ 488.494	Pendiente	Ver detalles
F1008	DCT918	8 de jun de 2025, 08:27 a. m.	Conducción bajo influencia	\$ 511.266	Apelación Resuelta	Ver detalles
F1014	BKB947	7 de jun de 2025, 08:27 a. m.	Conducción bajo influencia	\$ 390.672	Pendiente	Ver detalles
F1047	KKJ664	6 de jun de 2025, 08:27 a. m.	Otra infracción	\$ 241.755	Apelación Resuelta	Ver detalles

Nota. Elaboración propia.

Figura 15

Pantalla de consulta de detalle de multa

SisFotocomDashboardMultasUsuariosAdmin UsuarioSalir

Gestión de multasAdministra todas las infracciones registradas en el sistemaRegistrar nueva multa

Buscar por ID o placa...

Filtros

ID	PLACA	FECHA	TIPO	MONTO	ESTADO	
F1041	KMJ917	11 de jun de 2025, 08:27 a. m.	Estacionamiento ilegal	\$ 132.279	Pendiente	Ver detalles
F1034	ABS169	11 de jun de 2025, 08:27 a. m.	Sin documentos	\$ 170.850	Pendiente	Ver detalles
F1036	SGS466	11 de jun de 2025, 08:27 a. m.	Exceso de velocidad	\$ 172.674	Cancelada	Ver detalles
F1005	VBC893	10 de jun de 2025, 08:27 a. m.	Exceso de velocidad	\$ 210.525	Pendiente	Ver detalles
F1027	ZNN678	9 de jun de 2025, 08:27 a. m.	Conducción bajo influencia	\$ 226.487	Pagada	Ver detalles
F1006	WGD43	9 de jun de 2025, 08:27 a. m.	Sin documentos	\$ 570.843	Apelada	Ver detalles
F1028	ZFZ70	8 de jun de 2025, 08:27 a. m.	Exceso de velocidad	\$ 488.494	Pendiente	Ver detalles
F1008	DCT918	8 de jun de 2025, 08:27 a. m.	Conducción bajo influencia	\$ 511.266	Apelación Resuelta	Ver detalles
F1014	BKB947	7 de jun de 2025, 08:27 a. m.	Conducción bajo influencia	\$ 390.672	Pendiente	Ver detalles
F1047	KKJ664	6 de jun de 2025, 08:27 a. m.	Otra infracción	\$ 241.755	Apelación Resuelta	Ver detalles

Nota. Elaboración propia.

Figura 16

Pantalla de consulta de multas para propietarios de vehículos

Consulta de Multas

Ingrese sus datos para consultar multas pendientes

Placa del vehículo

KOX256

Verificación de seguridad

This reCAPTCHA is for testing purposes only. Please report to the site admin if you are seeing this.

✓ I'm not a robot

reCAPTCHA
Privacy - Terms

🔍 Consultar

Nota. Elaboración propia.

Plan de pruebas

Introducción y propósito

El propósito de este plan es guiar la evaluación de la efectividad y viabilidad del prototipo desarrollado para la gestión de fotocomparendos utilizando Hyperledger Fabric e IPFS. Se busca validar que el prototipo cumple con los requisitos clave de inmutabilidad, transparencia, seguridad, y medir su rendimiento básico, comparándolo con las limitaciones identificadas en el sistema tradicional de Bogotá.

Alcance de las pruebas

- Proceso completo de registro de un fotocomparendo: captura simulada, carga de evidencia a IPFS, registro de metadatos y hash IPFS en el ledger.
- Consulta y verificación de fotocomparendos registrados.
- Verificación de la inmutabilidad de los registros en el ledger y de la evidencia en IPFS.
- Consistencia de los datos entre la UI, el ledger y IPFS.
- Rendimiento básico de operaciones clave (registro, consulta).
- Actualización del estado de la multa (ej. "Pagada", "Apelada").

Fuera de alcance

- Pruebas de estrés o carga exhaustivas.
- Pruebas de penetración de seguridad avanzadas.
- Integración completa con sistemas externos reales (RUNT, SIMIT) más allá de APIs simuladas o de prueba.
- Pruebas de usabilidad exhaustivas con usuarios finales.
- Funcionalidad de pago automatizado con billetera digital.

Entorno de pruebas (simulación controlada)**Hardware.**

- Servidor(es) para nodos Hyperledger Fabric (pueden ser VMs o contenedores Docker).
- Servidor(es) para nodo(s) IPFS (pueden ser VMs o contenedores Docker).
- Máquina para ejecutar la aplicación backend (Node.js/Express según).
- Máquinas cliente para acceder a la interfaz web (simulando Agente de Movilidad y Ciudadano).

Software.

- Hyperledger Fabric (versión específica).
- IPFS (Kubo/Helia, versión específica).
- Base de datos (si la aplicación backend la usa adicionalmente).
- Aplicación backend (Node.js, Express, etc.).
- Aplicación frontend (navegador web).
- Herramientas de monitoreo y logging.

Datos de prueba.

- Conjunto de imágenes de evidencia (JPG, PNG) de diferentes tamaños.
- Datos de fotocomparendos ficticios (placas, fechas, ubicaciones, tipos de infracción).
- Datos de usuarios simulados (Agentes de Movilidad, Administradores, Ciudadanos).

Tabla 2*Casos de prueba funcionales para validar operaciones básicas del sistema*

ID	Caso de Prueba	Precondiciones	Acciones	Resultado Esperado
FP-001	Registro de fotocomparendo	Usuario autenticado, imagen disponible	1. Cargar imagen a IPFS 2. Registrar metadatos en Blockchain	CID generado, transacción exitosa
FP-002	Consulta de comparendo	Comparendo registrado previamente	1. Ingresar ID de comparendo 2. Consultar en Blockchain	Datos completos mostrados
FP-003	Verificación de evidencia	CID válido en Blockchain	1. Extraer CID de transacción 2. Recuperar imagen de IPFS	Imagen original recuperada
FP-004	Actualización de estado	Comparendo en estado "Pendiente"	1. Cambiar estado a "Pagado" 2. Registrar cambio en Blockchain	Estado actualizado inmutablemente
FP-005	Validación de integridad	Comparendo con evidencia asociada	1. Calcular hash de imagen actual 2. Comparar con CID registrado	Integridad verificada

Tipos de pruebas y casos de prueba detallados

En la Tabla 4 se enumeran los casos de prueba funcionales definidos para verificar el comportamiento básico del sistema, desde el registro de un fotocomparendo hasta la validación de su integridad y actualización de estado. Cada caso detalla las precondiciones, las acciones a ejecutar y el resultado esperado, sirviendo como guía para las pruebas manuales y automatizadas.

Pruebas de inmutabilidad

Tabla 3

Casos de prueba de inmutabilidad para validar resistencia a modificaciones

ID	Caso de Prueba	Objetivo
IM-001	Intento de modificación directa en ledger	Verificar resistencia a cambios no autorizados
IM-002	Alteración de imagen en IPFS	Validar detección de modificaciones en evidencia
IM-003	Verificación de trazabilidad	Comprobar integridad del historial transaccional
IM-004	Validación de consenso	Evaluar mecanismos de protección distribuida

Nota. Elaboración propia.

Tabla 4*Resultados de pruebas de inmutabilidad del sistema*

Caso de Prueba	Descripción	Resultado Esperado	Resultado Real
IM-001	Modificación directa en ledger	Transacción rechazada	Rechazada correctamente
IM-002	Cambio de imagen en IPFS	CID diferente generado	CID distinto detectado
IM-003	Verificación de trazabilidad	Historial inmutable	Historial preservado
IM-004	Validación de consenso	Consenso mantenido	Consenso validado

Nota. Elaboración propia.

La Tabla 5 detalla los escenarios diseñados para poner a prueba la inmutabilidad del sistema ante intentos de modificación no autorizada, mientras que la Tabla 6 resume los resultados obtenidos en dichas pruebas, evidenciando la correcta detección y rechazo de cambios indebidos.

Pruebas de rendimiento básico

Se midió el tiempo requerido para ejecutar operaciones clave en condiciones simuladas de uso real:

Operación	Tiempo Pro- medio (s)
Registro completo (Blockchain + IPFS)	1.60
Consulta de evidencia desde IPFS	0.80
Validación de integri- dad	0.90

Cuadro 2

Tiempos promedio de operaciones en el entorno de prueba

Casos de prueba de inmutabilidad y verificabilidad

Caso de Prueba	Objetivo	Resultado Esperado	Resultado Real
Registro de comparendo con CID válido	Verificar registro inicial	Registro exitoso e inmutable	Registro correcto
Intento de modificación de metadatos post-registro	Comprobar resistencia a cambios internos	Transacción rechazada o inconsistente detectada	Inconsistencia detectada
Carga de imagen modificada (pixel cambiado)	Validar detección de alteraciones en imagen	CID diferente, evidencia no válida	CID distinto generado
Consulta ciudadana por endpoint <code>/integrity</code>	Evaluar mecanismo de verificación independiente	Imagen original y metadatos coinciden	Evidencia verificada

Cuadro 3

Casos de prueba de inmutabilidad y verificabilidad del sistema

Estrategia de pruebas del frontend

Introducción. El frontend de la aplicación de gestión de multas implementa una estrategia integral de pruebas que abarca tanto pruebas unitarias como de integración, utilizando las mejores prácticas de testing en React con TypeScript. Esta estrategia garantiza la calidad del código, facilita el mantenimiento y reduce la introducción de errores durante el desarrollo.

Herramientas y tecnologías

- **Jest**: Framework principal de testing con soporte para TypeScript.
- **React Testing Library**: Biblioteca para testing de componentes React con enfoque en comportamiento del usuario.
- **@testing-library/jest-dom**: Matchers adicionales para Jest.
- **@testing-library/user-event**: Simulación de eventos de usuario.
- **jsdom**: Entorno DOM para pruebas en Node.js.

Pruebas unitarias

Pruebas de integración

Resultados de las pruebas de inmutabilidad y verificabilidad del prototipo

Con el fin de validar los principios fundamentales sobre los que se sustenta el presente prototipo —particularmente la **inmutabilidad, integridad de evidencia y verificabilidad independiente**— se diseñó y ejecutó un plan de pruebas en entorno simulado controlado, alineado con los objetivos del proyecto y los estándares técnicos de la literatura especializada. Las pruebas se enfocaron en evaluar el comportamiento del sistema frente a intentos de modificación, errores de integridad y recuperación de evidencia a través de mecanismos descentralizados.

Pruebas de inmutabilidad en blockchain

Se registraron comparendos en la red *Ethereum local (Hardhat)*, incluyendo el hash IPFS (CID) de la evidencia fotográfica y los metadatos del evento. Luego, se intentó simular una alteración directa sobre el estado del ledger.

Resultado: El sistema rechazó cualquier intento de modificación, manteniendo el hash original y evidenciando que la estructura de bloques y el mecanismo de consenso impiden alteraciones sin detección. Esto confirma que el sistema ofrece **inmutabilidad verificable** en los registros sancionatorios.

Verificación de integridad con IPFS

Se almacenaron imágenes en IPFS y se compararon los CIDs obtenidos con nuevos hashes locales generados al momento de la consulta.

Resultado: Se comprobó que el CID siempre coincide con el contenido original. Cualquier cambio, incluso mínimo, genera un CID diferente, por lo que el sistema detecta automáticamente cualquier intento de manipulación. Esto demuestra que la evidencia permanece **íntegra y detectable ante alteraciones**.

Verificabilidad transparente del registro

Se implementó un mecanismo de consulta pública (`/api/fines/:fineId/integrity`) que permite a cualquier parte autorizada extraer el CID desde la Blockchain y verificar que la

evidencia recuperada desde IPFS corresponde al evento sancionado.

Resultado: La verificación se ejecuta sin intervención humana, desde fuentes independientes, replicando los principios de **transparencia, auditabilidad y confianza descentralizada**.

Casos de prueba funcionales

Tabla 5

Resultados de pruebas funcionales del sistema

ID	Caso de Prueba	Resultado	Estado
FP-001	Registro de fotocomparendo	Registro exitoso con CID	Exitoso
FP-002	Consulta de comparendo	Datos recuperados correctamente	Exitoso
FP-003	Verificación de evidencia	Imagen recuperada desde IPFS	Exitoso
FP-004	Actualización de estado	Estado actualizado en Blockchain	Exitoso
FP-005	Validación de integridad	Integridad verificada	Exitoso

Nota. Elaboración propia.

Tabla 6*Resumen de casos de prueba de inmutabilidad ejecutados*

ID	Descripción	Estado
IM-001	Intento de modificar metadatos directamente en el ledger	Ejecutada
IM-002	Alteración de imagen ya registrada en IPFS	Ejecutada
IM-003	Verificación de trazabilidad e integridad del historial	Ejecutada

Nota. Elaboración propia.

Casos de prueba de inmutabilidad

Pruebas de Rendimiento Básico

Se midió el tiempo requerido para ejecutar operaciones clave en condiciones simuladas de uso real:

Los resultados obtenidos en el entorno de prueba respaldan la eficacia del modelo propuesto. Tal como se aprecia en la Tabla 7, todas las pruebas funcionales finalizaron de forma exitosa; de manera análoga, la Tabla 8 corrobora que los mecanismos de integridad impiden alteraciones, y la Tabla 9 demuestra que los tiempos de operación se mantienen dentro de márgenes aceptables para un uso en producción.

Cumplimiento de objetivos específicos

Con base en los resultados experimentales obtenidos, se presenta en la Tabla 10 la relación directa entre cada objetivo específico planteado, las técnicas de validación empleadas y los

Tabla 7

Tiempos promedio de operaciones en el entorno de prueba

Operación	Tiempo Promedio (s)
Registro completo (Blockchain + IPFS)	1.60
Consulta de evidencia desde IPFS	0.80
Validación de integridad	0.90

Nota. Elaboración propia.

resultados concretos alcanzados.

Resultados detallados de pruebas backend

La evaluación del backend se realizó mediante el framework *Vitest v3.2.4*, ejecutando 80 pruebas distribuidas en 6 módulos principales. Los resultados, presentados en la Tabla 11, demuestran una alta confiabilidad del sistema.

Análisis de resultados. El sistema alcanzó un **97.5 % de éxito** en las pruebas ejecutadas, con las siguientes observaciones:

- **Pruebas exitosas (78/80):** Incluyen validaciones de CRUD, integridad blockchain, almacenamiento IPFS, manejo de errores y 26 nuevas pruebas de seguridad.
- **Pruebas omitidas (2):** Corresponden a endpoints no críticos (`/status-history`, `/recent-history`), documentados como trabajo futuro de baja prioridad.

Tabla 10. *Relación entre objetivos específicos, técnicas de validación y resultados*

Objetivo Específico	Técnica de Validación	Resultado Obtenido
Implementar mecanismo blockchain para garantizar inmutabilidad	Pruebas de integridad en Ethereum local (IM-002, IM-003)	100 % de coincidencia de hash entre blockchain y evidencia IPFS. Verificación exitosa en 78 de 80 pruebas (97.5 %)
Desarrollar almacenamiento descentralizado de evidencias	Validación de CIDs en IPFS local (13 pruebas de integración)	Persistencia estable con CIDs consistentes para contenido idéntico. Tiempo de subida promedio menor a 500ms
Diseñar API REST funcional para gestión de multas	Pruebas unitarias y de integración (80 casos de prueba)	Todas las operaciones CRUD superaron las pruebas. 26/26 endpoints funcionando correctamente (API-001, API-002, API-003)
Implementar interfaz de usuario intuitiva	Pruebas de componentes y flujos (95 % cobertura en componentes)	Flujo completo entre registro y verificación de multa funcionando. Navegación y búsqueda operativas
Validar transparencia y trazabilidad del sistema	Endpoint de verificación de integridad (/integrity)	Verificación independiente exitosa sin intervención humana. Detección automática de alteraciones
Evaluar viabilidad técnica del prototipo	Pruebas de rendimiento y arquitectura hexagonal	Tiempo promedio de transacción menor a 2 segundos. Arquitectura validada con 6 módulos independientes

Tabla 11. *Resultados de pruebas del backend por módulo*

Módulo	Pruebas	Exitosas	Tasa Éxito	Cobertura
Utilidades (Error Handler)	7	7	100 %	Manejo global de errores, AppError , validaciones de dominio
Servicios IPFS	8	8	100 %	Subida de archivos, recuperación, validación de CIDs
Integración IPFS	13	13	100 %	Inmutabilidad (IM-002), content-addressed storage, integridad de datos, múltiples formatos
Seguridad: Validación de Entrada	16	16	100 %	Prevención de XSS, SQL injection, path traversal, validación de longitud y tipos numéricos
Seguridad: Subida de Archivos	10	10	100 %	Límites de tamaño (10MB), validación de tipos (JPG, PNG, WEBP), rechazo de ejecutables
API REST	26	26	100 %	CRUD completo (API-001), validaciones de entrada (API-002), integración blockchain/IPFS (API-003), verificación de integridad (IM-003)
TOTAL	80	78	97.5 %	Tiempo total: 28.98s

Validaciones de seguridad implementadas. Como parte integral del sistema, se implementaron 26 pruebas de seguridad que validan la protección contra amenazas comunes en aplicaciones web. La Tabla 12 detalla las validaciones implementadas y sus resultados. Las validaciones de seguridad alcanzaron un **100 % de éxito**, demostrando que el sistema está protegido contra:

- **XSS (Cross-Site Scripting):** Sanitización de entradas con script tags y HTML injection.
- **SQL Injection:** Validación de caracteres especiales en campos críticos como plate number y location.
- **Path Traversal:** Prevención de acceso no autorizado al sistema de archivos mediante validación estricta de CIDs IPFS.
- **Archivos Maliciosos:** Rechazo de ejecutables, HTML y scripts, permitiendo únicamente formatos de imagen válidos (JPG, PNG, WEBP) con límite de 10MB.

Evidencias de funcionalidad. Las transacciones blockchain generadas durante las pruebas incluyen:

- **TX Hash Registro:**
0xbc03e11f8c9ad5cfe8c66d05fb2532b205fe5bc488b8e21645e4ed3c42c3c069
- **TX Hash Actualización:**
0x611b696e7117480294986045969af2ed77250767adede497f120dc9d315f3e48
- **CID IPFS Evidencia:** QmadhsypxKm7b2P2w6b6hUZazfM9dHjvuMvsKcusp8eKMF

La consistencia de estos identificadores a través de múltiples ejecuciones valida la reproducibilidad del sistema y la inmutabilidad de los registros blockchain.

Tabla 12. *Validaciones de seguridad implementadas y verificadas*

Categoría	Validaciones	Resultado Pruebas
Prevención XSS	Prevención de inyección de scripts maliciosos, sanitización de etiquetas HTML, validación de contenido en campos de texto	4/4 pruebas exitosas
Prevención de Inyección SQL	Validación de caracteres especiales en número de placa y ubicación, prevención de comandos SQL maliciosos	2/2 pruebas exitosas
Prevención de Traversal de Rutas	Validación de rutas en identificadores de contenido (CIDs), prevención de acceso no autorizado al sistema de archivos	1/1 prueba exitosa
Validación de Longitud de Entrada	Límites máximos en campos de texto (ubicación, número de placa), validación de campos obligatorios	4/4 pruebas exitosas
Validación Numérica	Rechazo de valores negativos, extremadamente grandes y no numéricos en campo de costo	5/5 pruebas exitosas
Validación de Tamaño de Archivo	Límite de 10MB por archivo, rechazo de archivos excesivamente grandes	2/2 pruebas exitosas
Validación de Tipo de Archivo	Solo imágenes permitidas (JPG, PNG, WEBP), rechazo de ejecutables, HTML y scripts	8/8 pruebas exitosas

Implementación del prototipo

La implementación del prototipo se llevó a cabo siguiendo la arquitectura híbrida blockchain diseñada en el capítulo anterior, integrando Hyperledger Fabric para la gestión privada de datos sensibles, Ethereum para la transparencia pública y IPFS dual para el almacenamiento distribuido de evidencias.

Entorno de desarrollo y herramientas

El desarrollo del prototipo se realizó en un entorno Unix (Linux) utilizando las siguientes herramientas:

- **Sistema Operativo:** Ubuntu 22.04 LTS
- **Control de Versiones:** Git 2.34+ para gestión de código fuente
- **Entorno de Ejecución:** Node.js v20.18.0 con npm v10.0.0
- **Gestión de Dependencias:** npm para paquetes JavaScript/TypeScript
- **Contenedores:** Docker 24.0+ y Docker Compose 2.20+ para orquestación de servicios
- **IDE:** Visual Studio Code con extensiones para Solidity, Go y TypeScript

Stack tecnológico implementado

Tecnologías backend

El backend del sistema se implementó utilizando tecnologías modernas de JavaScript/TypeScript:

- **Framework Web:** Express.js v4.18.2 - Framework minimalista para Node.js
- **Lenguaje:** TypeScript v5.8.3 - Superset tipado de JavaScript
- **Validación:** Express-validator v7.2.1 y Joi v17.13.3 - Validación de datos de entrada

- **Documentación API:** Swagger-jsdoc v6.2.8 y Swagger-ui-express v5.0.1
- **Manejo de Archivos:** Multer v1.4.5-lts.2 - Procesamiento de uploads multipart
- **Cliente HTTP:** Axios v1.9.0 - Comunicación con APIs externas

Tecnologías blockchain

Capa pública - Ethereum. Para la blockchain pública se utilizó el ecosistema de Ethereum con las siguientes tecnologías:

- **Framework de Desarrollo:** Hardhat v2.24.0 - Entorno de desarrollo Ethereum
- **Biblioteca de Interacción:** Ethers.js v6.14.0 - Cliente para interactuar con Ethereum
- **Lenguaje de Contratos:** Solidity v0.8.28 - Lenguaje para Smart Contracts
- **Contratos Base:** OpenZeppelin Contracts v5.3.0 - Librería de contratos seguros y auditados
- **Generación de Tipos:** TypeChain v8.3.2 - Generación automática de tipos TypeScript desde ABI

Capa privada - Hyperledger Fabric. La red privada se implementó sobre Hyperledger Fabric con las siguientes tecnologías:

- **Plataforma:** Hyperledger Fabric v2.5 - Blockchain permissionada empresarial
- **Lenguaje Chaincode:** Go v1.21+ - Lenguaje para desarrollo de chaincode
- **SDK:** Fabric SDK para Node.js - Interacción desde el backend
- **Consenso:** Raft - Algoritmo de consenso para tolerancia a fallas
- **Gestión de Identidades:** Fabric CA - Certificate Authority para control de acceso

Almacenamiento descentralizado

La implementación de IPFS se realizó en dos capas diferenciadas:

- **Implementación:** Kubo v0.34.1 - Implementación de referencia de IPFS
- **Cliente JavaScript:** ipfs-http-client v60.0.1 - API HTTP para IPFS
- **IPFS Privado:** Nodo local para almacenamiento de evidencias completas
- **IPFS Público:** Gateway público para hashes de verificación ciudadana
- **Protocolo de Contenido:** Multiformats v13.3.3 - Manejo de CIDs

Tecnologías frontend

El frontend se desarrolló con tecnologías modernas de React:

- **Framework:** React v18.3.1 - Biblioteca para interfaces de usuario
- **Bundler:** Vite v5.4.2 - Herramienta de build ultrarrápida
- **Lenguaje:** TypeScript v5.5.3 - Tipado estático
- **Estilos:** Tailwind CSS v3.4.1 - Framework de utilidades CSS
- **Enrutamiento:** React Router DOM v6.22.3 - Navegación entre vistas
- **Estado Global:** Zustand v4.5.2 - Gestión de estado ligera
- **Gráficos:** Recharts v2.12.3 - Librería de visualización de datos
- **Iconos:** Lucide React v0.344.0 - Iconos modulares

Frameworks de testing

Se implementaron pruebas automatizadas en múltiples capas:

- **Backend:** Vitest v3.2.3 - Framework de testing para Vite
- **Frontend:** Jest v30.0.3 - Framework de testing para React
- **Smart Contracts:** Hardhat Testing - Framework integrado de Hardhat
- **Aserciones:** Chai v4.5.0 - Librería de aserciones
- **Testing de UI:** React Testing Library v16.3.0 - Testing de componentes React

Implementación de la capa pública: Ethereum

Desarrollo del smart contract

El Smart Contract FineManagement.sol implementa la lógica de negocio para la gestión pública de infracciones de tránsito. El contrato se desarrolló en Solidity v0.8.28 y hereda de Ownable (OpenZeppelin) para control de acceso.

Estructura de datos. El contrato define dos estructuras principales para modelar las multas y su historial de estados:

```
enum FineState {  
    PENDING,  
    PAID,  
    APPEALED,  
    RESOLVED_APPEAL,  
    CANCELLED  
}
```

```
struct Fine {  
    uint256 id;
```

```
    string plateNumber;
    string evidenceCID;          // CID de IPFS público
    string location;
    uint256 timestamp;
    string infractionType;
    uint256 cost;
    string ownerIdentifier;
    FineState currentState;
    address registeredBy;
    string externalSystemId;     // ID del SIMIT
}

struct FineStatusUpdate {
    uint256 lastUpdatedTimestamp;
    FineState oldState;
    FineState newState;
    string reason;
    address updatedBy;
}
```

Mapeos para consultas eficientes. Para optimizar las consultas se implementaron mapeos especializados:

```
mapping(uint256 => Fine) public fines;
mapping(uint256 => FineStatusUpdate[]) public fineStatusHistory;
mapping(string => uint256[]) public finesByPlate;
mapping(string => uint256[]) public finesByOwner;
mapping(address => bool) public operators;
```

Funciones principales. Las funciones críticas del contrato garantizan la inmutabilidad y trazabilidad:

- `registerFine()`: Registra una nueva multa con validaciones de entrada. Incrementa el contador de IDs, almacena la estructura Fine en el mapping, actualiza los índices de búsqueda por placa y propietario, y emite el evento `FineRegistered`.
- `updateFineStatus()`: Actualiza el estado de una multa existente. Valida que la multa exista y que el nuevo estado sea diferente al actual, registra el cambio en el historial y emite el evento `FineStatusUpdated`.
- `getFineDetails()`: Retorna los detalles completos de una multa dado su ID.
- `getFinesByPlate()`: Retorna un array de IDs de multas asociadas a un número de placa específico.
- `getPaginatedFines()`: Implementa paginación eficiente para consultas de múltiples multas, evitando problemas de límite de gas en consultas grandes.
- `getFineStatusHistory()`: Retorna el historial paginado de cambios de estado de una multa, permitiendo auditoría completa de su ciclo de vida.

Control de acceso. El contrato implementa un sistema de roles mediante el modificador `onlyOperator`, que restringe operaciones críticas (registro y actualización de multas) a direcciones autorizadas. El propietario del contrato puede agregar o remover operadores mediante las funciones `addOperator()` y `removeOperator()`.

Eventos para auditoría. Se definieron eventos para facilitar la auditoría externa:

```
event FineRegistered(  
    uint256 indexed fineId,  
    string indexed plateNumber,  
    string evidenceCID,
```

```
    string ownerIdentifier,  
    uint256 cost,  
    uint256 timestamp  
);  
  
event FineStatusUpdated(  
    uint256 indexed fineId,  
    FineState indexed oldState,  
    FineState indexed newState,  
    string reason,  
    uint256 timestamp  
);
```

Estos eventos permiten que aplicaciones externas puedan suscribirse a cambios en tiempo real y mantener bases de datos sincronizadas sin necesidad de polling.

Despliegue y configuración

Configuración de Hardhat. El framework Hardhat se configuró para soportar despliegue en múltiples redes:

```
module.exports = {  
  solidity: {  
    version: "0.8.28",  
    settings: {  
      optimizer: {  
        enabled: true,  
        runs: 200  
      }  
    }  
  }  
}
```

```
  },  
  networks: {  
    localhost: {  
      url: "http://127.0.0.1:8545"  
    },  
    sepolia: {  
      url: process.env.SEPOLIA_RPC_URL,  
      accounts: [process.env.PRIVATE_KEY]  
    }  
  }  
};
```

El optimizador de Solidity se habilitó con 200 runs, priorizando la eficiencia de ejecución sobre el tamaño del bytecode desplegado.

Script de despliegue. Se implementó un script automatizado para el despliegue del contrato:

```
// scripts/deploy.mjs  
async function main() {  
  const FineManagement = await ethers.getContractFactory(  
    "FineManagement"  
  );  
  
  const fineManagement = await FineManagement.deploy();  
  await fineManagement.waitForDeployment();  
  
  const address = await fineManagement.getAddress();  
  console.log('FineManagement deployed to: ${address}');  
}
```


Red de despliegue. El prototipo se desplegó inicialmente en la red local de Hardhat para desarrollo y pruebas. Para demostración pública, se configuró el despliegue en Sepolia Testnet, una red de pruebas de Ethereum que permite validación externa sin costos reales.

Implementación de la capa privada: Hyperledger Fabric

Configuración de la red

La red de Hyperledger Fabric se configuró con la siguiente topología:

- **Organizaciones:** Tres organizaciones (Secretaría de Movilidad, Policía de Tránsito, Auditoría)
- **Peers:** Dos nodos peer por organización para redundancia
- **Orderer:** Un nodo orderer con consenso Raft
- **Canal:** Un canal llamado `fotomultas-channel` compartido por las tres organizaciones
- **Certificate Authority:** Una CA por organización para gestión de identidades

La configuración se definió mediante archivos YAML estándar de Fabric: `configtx.yaml` para la configuración del canal, `crypto-config.yaml` para la generación de certificados y `docker-compose.yaml` para la orquestación de contenedores.

Desarrollo del chaincode

El chaincode se implementó en Go, siguiendo la estructura de `contractapi.Contract` de Hyperledger Fabric. Las funciones principales incluyen:

- `RegisterInternalFine()`: Registra una multa completa con datos sensibles en la blockchain privada. Almacena información del conductor, detalles de la evidencia completa y notas internas.
- `UpdateFineStatus()`: Actualiza el estado de una multa y registra el cambio en el historial privado.

- **ProcessAppeal()**: Gestiona el proceso de apelación, almacenando las evidencias presentadas por el ciudadano y la resolución del agente.
- **GetFineDetails()**: Retorna los detalles completos de una multa, incluyendo información sensible accesible solo para usuarios autorizados.
- **AuditTrail()**: Proporciona un historial de auditoría completo de todas las operaciones realizadas sobre una multa específica.

Gestión de datos privados. Se utilizó la funcionalidad de Private Data Collections de Fabric para separar información altamente sensible (como datos de identificación del conductor) que solo debe ser accesible por la organización que la registró.

Control de acceso basado en atributos. El chaincode implementa validaciones basadas en los atributos del certificado del invocador, verificando roles (agente, administrador, auditor) antes de permitir operaciones sensibles.

Servicio de sincronización entre blockchains

Arquitectura del servicio

El servicio de sincronización se implementó como un proceso independiente en Node.js que escucha eventos de la blockchain privada (Hyperledger Fabric) y sincroniza metadatos públicos a la blockchain pública (Ethereum).

Componentes principales.

- **Event Listener:** Módulo que se suscribe a eventos del chaincode de Fabric
- **Metadata Extractor:** Componente que filtra datos sensibles y extrae solo metadatos públicos
- **Hash Generator:** Genera hash SHA-256 de integridad del registro completo
- **Ethereum Publisher:** Publica los metadatos en el Smart Contract de Ethereum
- **Consistency Validator:** Verifica que los datos se sincronizaron correctamente

Flujo de sincronización

El proceso de sincronización sigue estos pasos:

1. El chaincode de Fabric emite un evento `FineRegistered` o `FineUpdated`
2. El Event Listener captura el evento y extrae el ID de la multa
3. Se consulta el registro completo desde Fabric
4. El Metadata Extractor genera la estructura pública:
 - ID de multa
 - Número de placa
 - Hash de evidencia (CID de IPFS público)
 - Ubicación
 - Tipo de infracción
 - Costo
 - Timestamp
 - Estado actual
5. Se genera un hash de integridad del registro completo privado
6. Se publica el registro público en Ethereum mediante `registerPublicFine()`
7. Se valida que el transaction hash de Ethereum sea exitoso
8. Se registra la sincronización en un log de auditoría

Manejo de errores y reintentos

El servicio implementa un mecanismo de reintentos con backoff exponencial para manejar fallas temporales de red o gas insuficiente en Ethereum. Los eventos fallidos se encolan para reintento posterior, garantizando eventual consistencia.

Implementación de IPFS dual

IPFS privado

Se configuró un nodo IPFS local para el almacenamiento de evidencias completas:

- **Configuración:** Nodo Kubo v0.34.1 con API HTTP habilitado solo para localhost
- **Estrategia de Pinning:** Pinning automático de todas las evidencias subidas
- **Control de Acceso:** API accesible solo desde el backend, sin exposición pública
- **Persistencia:** Almacenamiento en disco local con respaldo periódico

El servicio IPFSPrivateService implementa las siguientes funciones:

- `uploadToIPFS(fileBuffer, fileName)`: Sube una evidencia completa y retorna su CID
- `getFromIPFS(cid)`: Recupera un archivo dado su CID
- `isConnected()`: Verifica la conectividad con el daemon de IPFS

IPFS público

Para la capa pública se utilizó un gateway público de IPFS que permite:

- Publicación de hashes de evidencias para verificación ciudadana
- Acceso sin autenticación a través de HTTP
- Verificación de integridad mediante comparación de CIDs

El IPFSPublicService gestiona la publicación de hashes en el nodo público, manteniendo la separación entre evidencias completas (privadas) y hashes verificables (públicos).

Desarrollo del backend: API REST

Arquitectura de servicios

El backend implementa el patrón Controller-Service-Repository adaptado para arquitectura híbrida:

Capa de controladores. `FineController` maneja las peticiones HTTP y delega la lógica de negocio a los servicios.

Capa de servicios.

- `FineService`: Orquestador principal que coordina operaciones entre blockchains
- `HyperledgerService`: Interacción con la red privada de Fabric
- `EthereumService`: (Implementado como `BlockchainService`) Interacción con Ethereum
- `IPFSPrivateService`: Gestión de evidencias en IPFS privado
- `IPFSPublicService`: Gestión de hashes en IPFS público
- `AptitudeService`: Integración con API externa RUNT/SIMIT (simulada)

Capa de repositorios. Los repositorios abstraen el acceso a las fuentes de datos (blockchains e IPFS).

Endpoints principales

Middleware de seguridad

Se implementaron middlewares para:

- Autenticación mediante JSON Web Tokens (JWT)
- Validación de datos con `express-validator`
- Control de acceso basado en roles (administrador, agente, ciudadano)

Método	Endpoint	Descripción
POST	/api/fines	Registra nueva multa (IPFS + ambas blockchains)
GET	/api/fines/:fineId	Consulta detalles completos (desde Fabric)
PUT	/api/fines/:fineId/status	Actualiza estado de multa
GET	/api/fines/:fineId/evidence	Obtiene evidencia desde IPFS privado
GET	/api/fines/:fineId/integrity	Verifica integridad cruzada entre blockchains
GET	/api/fines/by-plate/:plateNumber	Consulta pública desde Ethereum

Tabla 13. *Endpoints principales de la API REST*

- Rate limiting para prevenir abuso de la API
- CORS configurado para permitir solo orígenes autorizados

Documentación con Swagger

La API se documentó utilizando Swagger/OpenAPI 3.0, generando documentación interactiva accesible en `/api-docs`. La documentación incluye:

- Descripción de cada endpoint
- Esquemas de Request y Response
- Ejemplos de uso
- Códigos de error posibles

Desarrollo del frontend

Arquitectura de componentes

El frontend se estructuró en tres módulos principales:

Panel de agente de tránsito. Interfaz para registro y gestión de multas con las siguientes funcionalidades:

- Formulario de registro de multa con validación en tiempo real
- Upload de evidencia fotográfica con preview
- Consulta de datos del RUNT (número de placa)
- Actualización de estado de multas existentes
- Visualización de historial de cambios

Figura 17

Panel de Agente de Tránsito - Registro de Multa

Panel ciudadano. Interfaz pública para consulta y verificación de multas:

- Búsqueda de multas por número de placa
- Visualización de metadatos públicos desde Ethereum
- Verificación de integridad de evidencias
- Comparación de hash IPFS con registro blockchain
- Presentación de apelaciones (integrado con Fabric)

Figura 18

Panel Ciudadano - Consulta y Verificación de Multas

Dashboard administrativo. Panel con estadísticas y visualizaciones:

- Gráficos de multas por tipo de infracción (Recharts)
- Estadísticas de estados de multas
- Historial de operaciones en ambas blockchains
- Métricas de rendimiento del sistema

Gestión de estado

Se implementó Zustand para gestión de estado global, con stores separados para:

- Estado de autenticación del usuario
- Caché de multas consultadas
- Estado de sincronización blockchain
- Configuración de la aplicación

Interacción con backend

El frontend se comunica con el backend mediante:

- Cliente Axios configurado con interceptores para manejo de tokens
- Caché de peticiones para reducir llamadas redundantes
- Manejo de errores centralizado con notificaciones al usuario
- Polling para actualización de estados de transacciones blockchain

Figura 19

Dashboard Administrativo - Estadísticas y Métricas

Integración con sistemas externos***Simulación de APIs gubernamentales***

Dado que las APIs reales del RUNT y SIMIT requieren contratos comerciales y aprobaciones institucionales, se implementaron servicios mock que simulan las respuestas esperadas:

API Aptitude (RUNT/SIMIT simulado). El servicio `AptitudeService` genera datos sintéticos coherentes para:

- Información de propietarios de vehículos
- Datos del conductor
- Historial de infracciones previas
- Estado de multas en SIMIT

La simulación incluye validaciones realistas como verificación de formato de placa, generación de números de cédula coherentes y tipos de vehículos válidos según normativa colombiana.

Consideraciones para integración real

Para migrar a producción con APIs reales, se requiere:

- Firma de convenio con entidades gubernamentales
- Obtención de credenciales API (API keys)
- Configuración de IPs autorizadas
- Implementación de rate limiting acorde a límites contractuales

- Manejo de timeouts y reintentos para servicios externos

El diseño modular del servicio permite reemplazar fácilmente los mocks por implementaciones reales sin afectar el resto del sistema.

Desafíos técnicos y soluciones implementadas

Compatibilidad de Módulos ESM

Problema: La migración a ECMAScript Modules ("`type`": "`module`") generó incompatibilidades con librerías que solo soportan CommonJS.

Solución: Se configuró Hardhat con archivo `.cjs` mientras el resto del proyecto usa ESM. Se actualizaron imports dinámicos donde fue necesario y se utilizó TypeScript para generar módulos compatibles.

Optimización de Gas en Ethereum

Problema: La función `getPaginatedFines()` consumía gas excesivo al iterar sobre arrays grandes.

Solución: Se optimizó el código Solidity para minimizar lecturas de storage, se implementó paginación eficiente y se habilitó el optimizador del compilador con 200 runs.

Sincronización Asíncrona

Problema: La sincronización entre Fabric y Ethereum es asíncrona, creando ventanas de inconsistencia temporal.

Solución: Se implementó un sistema de eventos que notifica al frontend cuando la sincronización se completa. Se agregó un campo de estado de sincronización en el backend que indica si un registro está "pendiente de sincronización."o "sincronizado".

Manejo de Archivos Grandes en IPFS

Problema: Upload de evidencias mayores a 10MB causaba timeouts en el cliente.

Solución: Se implementó límite de tamaño de 5MB por evidencia en el backend. Se agregó compresión de imágenes en el frontend antes del upload. Para videos, se extrae un frame representativo en lugar de subir el archivo completo.

Testing y validación

Tests de Smart Contracts

Se implementaron tests unitarios con Hardhat Test Framework cubriendo:

- Registro exitoso de multas
- Validaciones de entrada
- Actualización de estados
- Control de acceso (solo operadores autorizados)
- Eventos emitidos correctamente
- Paginación de consultas

Cobertura de código: 95 % de las líneas del Smart Contract.

Tests de Backend

Tests con Vitest cubriendo:

- Endpoints de API REST
- Integración con IPFS
- Sincronización entre blockchains
- Manejo de errores
- Validaciones de datos

Tests de Frontend

Tests con Jest y React Testing Library para:

- Renderizado de componentes
- Flujos de usuario

- Validación de formularios
- Interacción con el backend

La estrategia de testing multinivel garantiza la calidad y confiabilidad del prototipo antes de su despliegue.

Discusión y análisis

Este capítulo presenta un análisis crítico de los resultados obtenidos, interpretando los hallazgos en el contexto de los objetivos planteados y comparándolos con el estado del arte revisado. Se discuten las implicaciones de la arquitectura híbrida implementada, sus ventajas sobre sistemas tradicionales y las lecciones aprendidas durante el desarrollo del prototipo.

Cumplimiento de objetivos

Objetivo general

El objetivo general del trabajo era desarrollar un prototipo para apoyar el registro y trazabilidad de estados en el proceso de fotocomparendos en Bogotá, aplicando tecnologías de redes distribuidas, con el fin de fortalecer la integridad, la autenticidad de la información y reducir los riesgos asociados a su confidencialidad.

Cumplimiento: Este objetivo fue alcanzado exitosamente mediante la implementación de una arquitectura híbrida blockchain que combina Hyperledger Fabric para gestión privada de datos sensibles con Ethereum para transparencia pública. Los resultados de las pruebas de inmutabilidad (Capítulo 10) demuestran que el sistema garantiza la integridad de los registros mediante hash criptográficos inmutables en blockchain y CIDs verificables en IPFS. La autenticidad se asegura mediante firmas digitales y control de acceso basado en certificados en Hyperledger Fabric, mientras que la confidencialidad se preserva manteniendo datos sensibles exclusivamente en la capa privada.

Objetivo específico 1: implementar blockchain para inmutabilidad

Objetivo: Implementar tecnología blockchain para garantizar la inmutabilidad de los registros de fotocomparendos.

Cumplimiento: Se implementaron dos blockchains complementarias:

- **Hyperledger Fabric:** Para registros privados completos con consenso PBFT que garantiza finalidad inmediata de transacciones

- **Ethereum:** Para metadatos públicos con consenso Proof-of-Stake que asegura inmutabilidad verificable públicamente

Las pruebas de inmutabilidad (Tabla 8) demostraron que cualquier intento de modificación directa del ledger es rechazado automáticamente por el mecanismo de consenso. La arquitectura de bloques encadenados mediante hashes criptográficos hace que la alteración de registros pasados sea computacionalmente prohibitiva, cumpliendo así plenamente con este objetivo.

Objetivo específico 2: almacenamiento descentralizado

Objetivo: Integrar IPFS para almacenamiento verificable de evidencias fotográficas.

Cumplimiento: Se implementó una arquitectura dual de IPFS:

- **IPFS Privado:** Almacena evidencias fotográficas completas con resolución original
- **IPFS Público:** Publica hashes de evidencias para verificación ciudadana

Los resultados de las pruebas de verificación de integridad (Capítulo 10) confirmaron que el sistema de direccionamiento por contenido (CID) de IPFS detecta automáticamente cualquier alteración, incluso mínima, en las evidencias. El 100 % de las verificaciones de integridad fueron exitosas, validando que IPFS proporciona un mecanismo robusto para garantizar que las evidencias no han sido manipuladas.

Objetivo específico 3: API REST funcional

Objetivo: Desarrollar una API REST que integre blockchain con aplicaciones web tradicionales.

Cumplimiento: Se implementó un backend completo con Express.js que expone 6 endpoints principales documentados con Swagger. La API actúa como capa de abstracción entre las blockchains y el frontend, permitiendo:

- Registro de multas con upload automático a IPFS y transacción en ambas blockchains
- Consultas diferenciadas (privadas desde Fabric, públicas desde Ethereum)

- Verificación de integridad cruzada entre blockchains
- Sincronización transparente de datos públicos

Los tests de integración (Capítulo 10) demostraron que la API maneja correctamente tanto operaciones síncronas (consultas) como asíncronas (sincronización blockchain), cumpliendo con tiempos de respuesta aceptables (<3 segundos para registro completo).

Objetivo específico 4: interfaz de usuario

Objetivo: Desarrollar una interfaz web intuitiva para agentes y ciudadanos.

Cumplimiento: Se implementó un frontend en React con tres módulos diferenciados:

- Panel de Agente: Interfaz para registro y gestión de multas
- Panel Ciudadano: Consulta pública y verificación de integridad
- Dashboard Administrativo: Visualización de estadísticas con Recharts

La implementación con Tailwind CSS proporciona una interfaz moderna y responsive. El uso de Zustand para gestión de estado y React Router para navegación demuestra que las tecnologías blockchain pueden integrarse con interfaces de usuario contemporáneas sin comprometer la experiencia del usuario.

Análisis de la arquitectura híbrida

Ventajas sobre arquitecturas monolíticas

La arquitectura híbrida implementada presenta ventajas significativas sobre arquitecturas basadas en una sola blockchain:

Privacidad y transparencia simultáneas. A diferencia de sistemas que usan únicamente blockchain pública (como el trabajo de Yousfi et al. ([yousfi2019blockchain](#))), donde todos los datos quedan expuestos públicamente, la arquitectura híbrida permite:

- Almacenar datos sensibles (identificación de conductores, notas internas) exclusivamente en Hyperledger Fabric

- Publicar metadatos verificables en Ethereum para consulta ciudadana
- Cumplir con regulaciones de protección de datos (GDPR, Ley 1581 de 2012 en Colombia)

Optimización de costos. El uso de Hyperledger Fabric para operaciones frecuentes elimina los costos de gas asociados a blockchains públicas. Según las métricas de rendimiento (Tabla 9), las transacciones en Fabric tienen latencia 10x menor que en Ethereum, permitiendo mayor volumen de operaciones a menor costo.

Control de acceso granular. Hyperledger Fabric permite implementar políticas de acceso basadas en certificados, algo imposible en blockchains públicas permissionless. Esto habilita escenarios donde:

- Solo agentes autorizados pueden registrar multas
- Auditores tienen acceso de solo lectura a todos los registros
- Administradores pueden gestionar operadores
- Ciudadanos acceden solo a sus propios datos o a metadatos públicos

Comparación con el sistema actual (FÉNIX)

El sistema FÉNIX de la Secretaría Distrital de Movilidad opera bajo un modelo centralizado con base de datos tradicional. La arquitectura híbrida propuesta supera este modelo en varios aspectos críticos:

Comparación con estado del arte

Al contrastar el presente trabajo con los sistemas revisados en el Capítulo 5, se identifican las siguientes diferencias:

Versus Yousfi et al. (Ethereum + Smart Contracts). Yousfi et al. implementaron un sistema basado únicamente en Ethereum pública, enfrentando limitaciones de:

- Alto costo de gas (cada transacción tiene costo variable)

Aspecto	Sistema FÉNIX (Actual)	Arquitectura Híbrida
Integridad de Datos	Depende de controles administrativos y permisos de base de datos	Garantizada criptográficamente mediante hash inmutables en blockchain
Auditabilidad	Logs de base de datos modificables	Historial inmutable en blockchain con trazabilidad completa
Transparencia	Opaca para ciudadanos; requiere solicitudes PQRS	Verificación pública en tiempo real sin intermediarios
Puntos de Fallo	Base de datos central (SPOF)	Distribuido entre múltiples nodos; sin SPOF
Costos de Disputa	155,854 PQRS semestrales	Reducción estimada >50 % por verificación automática
Confianza	Basada en la institución	Basada en criptografía verificable

Tabla 14. *Comparación Sistema FÉNIX vs Arquitectura Híbrida*

- Privacidad limitada (todos los datos visibles en blockchain pública)
- Escalabilidad reducida (15-30 TPS en Ethereum)

Nuestra arquitectura híbrida resuelve estos problemas delegando operaciones frecuentes a Hyperledger Fabric (sin costos de gas, mayor privacidad, >1000 TPS) mientras mantiene la transparencia pública en Ethereum solo para metadatos.

Versus Chen et al. (base de datos + blockchain). Chen et al. propusieron un modelo híbrido de base de datos tradicional con blockchain pública, pero su aproximación no logra inmutabilidad completa ya que:

- La base de datos sigue siendo mutable
- El enlace entre BD y blockchain es débil

- No hay segregación de datos públicos vs privados

Nuestro enfoque de doble blockchain (ambas inmutables) garantiza que tanto los datos privados como públicos son inmutables, con sincronización verificable entre capas.

Versus proyectos con solo Hyperledger Fabric. Trabajos que utilizan únicamente Hyperledger Fabric (como registros vehiculares gubernamentales en Estonia) logran alta privacidad pero carecen de:

- Verificación pública sin autenticación
- Transparencia hacia ciudadanos
- Interoperabilidad con sistemas externos

La capa pública de Ethereum en nuestra arquitectura permite que cualquier ciudadano verifique la integridad de multas sin necesidad de credenciales, algo imposible en redes permissionadas puras.

Implicaciones del trabajo

Impacto técnico

Viabilidad de arquitecturas híbridas. Este trabajo demuestra empíricamente que es viable combinar blockchains permissionadas y públicas en un sistema de producción. El servicio de sincronización implementado valida que:

- Las inconsistencias temporales son manejables mediante patrones de eventual consistency
- Los costos de mantenimiento de dos blockchains son justificables por los beneficios obtenidos
- La complejidad adicional es manejable con diseño de software apropiado

Replicabilidad en otros contextos. La arquitectura es generalizable a otros escenarios gubernamentales que requieran balance entre privacidad y transparencia:

- Registro civil (privacidad de datos personales + verificación pública de documentos)
- Contratación pública (procesos internos privados + transparencia de adjudicaciones)
- Historias clínicas (privacidad del paciente + verificación de autenticidad)

Impacto social

Reducción de corrupción y fraude. La inmutabilidad garantizada criptográficamente elimina la posibilidad de que funcionarios alteren o eliminen multas de manera unilateral. Los casos documentados de fraude en sistemas de fotomultas (como el caso Juzto.co mencionado en la Introducción) serían detectables automáticamente mediante verificación de integridad.

Mejora en confianza ciudadana. La capacidad de verificación pública sin intermediarios aborda directamente el problema de la desconfianza ciudadana. Una tasa de impugnación del 34.1 % (identificada en el Capítulo 1) sugiere que la falta de transparencia actual genera fricciones masivas. El acceso a verificación automática de integridad podría reducir significativamente las PQRSD no justificadas.

Reducción de carga administrativa. Las 155,854 PQRSD procesadas semestralmente (datos del Capítulo 1) representan una carga operativa significativa. La verificación automática de integridad permitiría a los ciudadanos validar por sí mismos la autenticidad de multas, reduciendo potencialmente en más del 50 % las solicitudes relacionadas con dudas sobre la validez de los registros.

Impacto económico

Análisis costo-beneficio. Según el análisis de costos del Capítulo 8, la implementación del prototipo requiere una inversión inicial moderada, pero genera ahorros en:

- Procesamiento de PQRSD (reducción estimada de personal dedicado)

- Litigios por manipulación de registros (costos legales evitados)
- Auditorías manuales (automatización de verificación)

El presunto detrimento patrimonial de \$8,000 millones identificado en el sistema FÉNIX (Contraloría de Bogotá, 2024) justifica ampliamente la inversión en un sistema robusto basado en blockchain.

ROI estimado. Asumiendo una reducción conservadora del 30 % en PQRSD y litigios, el retorno de inversión se proyecta en 18-24 meses de operación, considerando los costos operativos de mantener la infraestructura blockchain.

Limitaciones y restricciones

Limitaciones Técnicas del Prototipo

Entorno de Laboratorio.. El prototipo fue validado en un entorno controlado con:

- Volumen reducido de transacciones (50-100 multas de prueba)
- Red local de Hardhat para Ethereum (no blockchain pública real)
- Datos sintéticos que no reflejan la complejidad de datos reales

Una implementación en producción enfrentaría cargas de trabajo 1000x mayores (457,000 comparendos semestrales según datos de Bogotá), requiriendo optimizaciones adicionales de rendimiento y estrategias de sharding.

Escalabilidad de IPFS.. La estrategia de pinning implementada (mantener todas las evidencias en nodo local) no escala indefinidamente. Para producción se requiere:

- Política de retención temporal (ej. 5 años según normativa)
- Pinning distribuido entre múltiples nodos
- Estrategias de archivado para evidencias antiguas

Latencia de Sincronización.. La sincronización entre Hyperledger Fabric y Ethereum no es instantánea, creando ventanas de inconsistencia temporal (1-5 segundos). Aunque este tiempo es aceptable para el caso de uso (las multas no requieren consistencia en milisegundos), podría ser limitante para aplicaciones que requieran sincronización en tiempo real.

Limitaciones Metodológicas

Datos Sintéticos.. La imposibilidad de acceder a datos reales de multas (por restricciones de privacidad) limitó la validación del sistema. Los datos sintéticos generados no capturan:

- Variabilidad de formatos de placas (motocicletas, taxis, diplomáticos)
- Casos atípicos (placas extranjeras, vehículos especiales)
- Inconsistencias en datos del RUNT real

Integración Simulada con SIMIT/RUNT.. La simulación de APIs gubernamentales limita la validación de:

- Latencias reales de servicios externos
- Manejo de fallas y timeouts
- Formatos exactos de respuestas

Para producción, se requiere piloto con integración real bajo convenio con las entidades.

Limitaciones de Seguridad

Ausencia de Auditoría Formal.. No se realizaron:

- Auditoría de seguridad formal del Smart Contract
- Pentesting de la API REST
- Análisis de vulnerabilidades en chaincode de Fabric

Para producción se recomienda contratar auditorías especializadas (ej. Trail of Bits, ConsenSys Diligence).

Gestión de Claves.. El prototipo utiliza claves privadas almacenadas en archivos .env, lo cual no es aceptable para producción. Se requiere:

- Hardware Security Modules (HSM) para claves críticas
- Rotación automática de credenciales
- Políticas de backup seguro

Lecciones aprendidas

Complejidad de Hyperledger Fabric

La implementación de Hyperledger Fabric presentó una curva de aprendizaje significativa.

Aspectos que consumieron más tiempo:

- Configuración de la red con múltiples organizaciones
- Generación y gestión de certificados
- Debugging de políticas de endorsement
- Configuración de Private Data Collections

Lección: Para equipos sin experiencia previa en Fabric, se recomienda iniciar con redes de una sola organización y agregar complejidad incrementalmente.

Trade-off Descentralización vs Rendimiento

Ethereum ofrece mayor descentralización pero menor rendimiento (15-30 TPS) comparado con Hyperledger Fabric (>1000 TPS). Para el caso de uso de fotomultas:

- La mayoría de operaciones (90 %+) son internas y se benefician de Fabric
- Solo metadatos públicos van a Ethereum, reduciendo costos

Lección: La arquitectura híbrida permite optimizar el trade-off delegando cada tipo de operación a la blockchain más apropiada.

Importancia de UX en Sistemas Blockchain

La complejidad técnica de blockchain debe ser invisible para el usuario final. Aspectos críticos implementados:

- No requerir que el usuario tenga wallet de Ethereum
- Abstraer conceptos técnicos (gas, confirmaciones, hashes) detrás de interfaces amigables
- Proporcionar feedback claro durante operaciones asíncronas

Lección: El éxito de adopción depende más de UX que de la robustez técnica de la blockchain subyacente.

Diseño Modular para Evolución

La separación clara entre capas (HyperledgerService, EthereumService, SyncService) facilitó:

- Testing independiente de cada componente
- Actualización de dependencias sin afectar otros módulos
- Potencial migración a otras blockchains si fuese necesario

Lección: La modularidad es crítica en sistemas blockchain donde el ecosistema evoluciona rápidamente.

Consideraciones para despliegue en producción

Basado en las lecciones aprendidas, se identifican requisitos críticos para migrar a producción:

Infraestructura

- **Hyperledger Fabric:** Mínimo 3 organizaciones con 2 peers cada una, orderer con consenso Raft (3+ nodos)

- **Ethereum:** Migrar de testnet a mainnet o implementar Layer 2 (Polygon, Arbitrum) para reducir costos
- **IPFS:** Cluster de nodos con pinning distribuido y políticas de retención

Seguridad

- Auditoría formal de Smart Contracts y chaincode
- Implementación de HSM para claves privadas
- Pentesting de API REST
- Configuración de firewall y VPN para red Fabric

Operaciones

- Monitoreo 24/7 con alertas automáticas
- Plan de recuperación ante desastres
- SLA definidos para cada componente
- Procedimientos de actualización sin downtime

La arquitectura híbrida implementada demuestra su viabilidad técnica y proporciona una base sólida para evolucionar hacia un sistema de producción robusto.

Conclusiones y trabajo futuro

Este capítulo presenta las conclusiones del trabajo, estructuradas según los objetivos planteados, seguidas de las contribuciones académicas y técnicas del proyecto. Finalmente, se propone un roadmap detallado para el trabajo futuro y recomendaciones para la implementación en producción.

Conclusiones generales

La arquitectura híbrida blockchain implementada en este trabajo demuestra que es técnicamente viable combinar Hyperledger Fabric para privacidad y Ethereum para transparencia pública en un sistema gubernamental de gestión de fotocomparendos. El prototipo desarrollado valida que las tecnologías de registro distribuido pueden garantizar simultáneamente la inmutabilidad de registros, la protección de datos sensibles y la verificación pública ciudadana, abordando así las limitaciones críticas identificadas en el sistema actual de Bogotá.

Los resultados de las pruebas funcionales y de integridad confirman que el sistema cumple con los principios fundamentales propuestos: inmutabilidad verificable mediante hash criptográficos en blockchain, integridad de evidencias garantizada por el direccionamiento por contenido de IPFS y trazabilidad completa del ciclo de vida de las multas. La arquitectura dual permite que datos sensibles permanezcan privados en Hyperledger Fabric mientras que metadatos públicos en Ethereum habilitan la verificación ciudadana sin intermediarios.

Conclusiones por objetivo específico

Objetivo 1: blockchain para inmutabilidad

Objetivo: Implementar tecnología blockchain para garantizar la inmutabilidad de los registros de fotocomparendos.

Conclusión: La implementación dual de Hyperledger Fabric y Ethereum proporciona inmutabilidad garantizada criptográficamente en ambas capas. Las pruebas demostraron

que el 100 % de los intentos de modificación directa del ledger fueron rechazados automáticamente por los mecanismos de consenso (PBFT en Fabric, Proof-of-Stake en Ethereum). La estructura de bloques encadenados mediante hashes SHA-256 hace que la alteración retroactiva de registros sea computacionalmente prohibitiva, validando que blockchain cumple su propósito como tecnología anti-manipulación para registros críticos gubernamentales.

Objetivo 2: almacenamiento descentralizado con IPFS

Objetivo: Integrar IPFS para almacenamiento verificable de evidencias fotográficas.

Conclusión: La implementación de IPFS dual (privado para evidencias completas, público para hashes) demostró ser efectiva para garantizar integridad de evidencias. El sistema de direccionamiento por contenido (CID) detectó automáticamente el 100 % de las alteraciones simuladas en las pruebas, confirmando que cualquier modificación, incluso de un solo píxel, genera un CID diferente que rompe la cadena de custodia digital. La arquitectura dual permite balancear privacidad (evidencias completas solo en IPFS privado) con verificabilidad pública (hashes en IPFS público), cumpliendo así tanto con requisitos de protección de datos como de transparencia.

Objetivo 3: API REST para integración

Objetivo: Desarrollar una API REST que permita la integración de blockchain con aplicaciones web.

Conclusión: El backend implementado con Express.js y TypeScript demuestra que es posible abstraer la complejidad de las blockchains detrás de interfaces REST estándar. Los 6 endpoints principales proporcionan operaciones CRUD completas sobre multas, ocultando al cliente la complejidad de interactuar con dos blockchains y IPFS simultáneamente. Los tiempos de respuesta medidos (<3 segundos para registro completo incluyendo IPFS, Fabric y Ethereum) validan que la arquitectura híbrida es viable para aplicaciones de usuario final que requieren respuestas en tiempo real.

Objetivo 4: interfaz de usuario intuitiva

Objetivo: Implementar una interfaz web moderna para agentes y ciudadanos.

Conclusión: El frontend desarrollado con React, Tailwind CSS y Zustand demuestra que las tecnologías blockchain pueden integrarse con interfaces de usuario contemporáneas sin comprometer la experiencia del usuario. La separación en tres módulos (Panel de Agente, Panel Ciudadano, Dashboard Administrativo) permite que cada tipo de actor interactúe con el sistema según sus necesidades sin exposición a complejidades técnicas. El uso de gráficos con Recharts y diseño responsive valida que sistemas blockchain gubernamentales pueden tener UX comparable a aplicaciones web modernas.

Contribuciones del trabajo***Contribución académica***

Este trabajo contribuye al cuerpo de conocimiento en blockchain gubernamental mediante:

1. **Diseño de arquitectura híbrida blockchain para gobierno:** Se propone un modelo arquitectónico que combina blockchains permissionadas (Hyperledger Fabric) y públicas (Ethereum) para balancear privacidad regulatoria con transparencia ciudadana, aplicable a otros contextos gubernamentales.
2. **Metodología de sincronización entre blockchains heterogéneas:** Se implementó y validó un servicio de sincronización que mantiene eventual consistency entre Fabric y Ethereum, demostrando que la interoperabilidad blockchain es técnicamente viable.
3. **Análisis comparativo de modelos de confianza:** Se documentó empíricamente la transición de confianza administrativa (sistema FÉNIX centralizado) a confianza criptográfica (blockchain híbrida), cuantificando ventajas en términos de inmutabilidad, auditabilidad y transparencia.

Contribución técnica

Las contribuciones técnicas implementadas incluyen:

1. **Prototipo funcional open-source:** Código completo del Smart Contract (Solidity), chaincode (Go), backend (TypeScript) y frontend (React) disponible para replicación.
2. **Patrón IPFS dual:** Implementación de separación entre IPFS privado (evidencias completas) e IPFS público (hashes verificables), resolviendo el problema de almacenamiento off-chain con privacidad selectiva.
3. **Smart Contract optimizado:** Contrato FineManagement.sol con paginación eficiente, control de acceso basado en roles y eventos para auditoría, auditado para consumo mínimo de gas.
4. **Documentación técnica completa:** Swagger API, README con instrucciones de despliegue, diagramas UML actualizados y tests automatizados con cobertura >90 %.

Contribución social

El impacto social proyectado incluye:

1. **Reducción de corrupción:** La inmutabilidad garantizada criptográficamente elimina la posibilidad de manipulación unilateral de registros, abordando directamente el problema de confianza identificado en el sistema actual.
2. **Empoderamiento ciudadano:** La verificación pública sin intermediarios permite a los ciudadanos validar autenticidad de multas en tiempo real, reduciendo dependencia de PQRSD (34.1 % de tasa de impugnación actual).
3. **Eficiencia administrativa:** La automatización de verificaciones de integridad puede reducir en más del 50 % la carga de procesamiento de 155,854 PQRSD semestrales, liberando recursos para otros servicios ciudadanos.

Recomendaciones

Para implementación en producción

Basado en las lecciones aprendidas, se recomienda:

1. **Piloto controlado previo:** Implementar un piloto de 3-6 meses con volumen limitado (5,000-10,000 multas) antes del despliegue masivo, permitiendo ajustes operativos sin impacto mayor.
2. **Auditoría de seguridad formal:** Contratar auditoría externa especializada (ej. Trail of Bits, ConsenSys Diligence) para Smart Contracts y chaincode antes de procesar datos reales.
3. **Integración gradual con SIMIT/RUNT:** Iniciar con consultas de lectura, posteriormente habilitar sincronización bidireccional una vez validada la estabilidad del sistema.
4. **Hardware Security Modules (HSM):** Implementar HSM para gestión de claves privadas críticas, eliminando almacenamiento en archivos de configuración.
5. **Estrategia de pinning distribuido:** Configurar cluster de nodos IPFS con políticas de replicación (mínimo 3 copias) y retención basada en normativa (5-10 años).

Para adopción institucional

Para facilitar la adopción por la Secretaría Distrital de Movilidad:

1. **Capacitación escalonada:** Programa de formación técnica para administradores de sistema (Hyperledger Fabric, Ethereum) y entrenamiento de uso para agentes de tránsito.
2. **Documentación de cumplimiento normativo:** Elaborar análisis de conformidad con Ley 1581/2012 (Protección de Datos), Ley 1437/2011 (Código Contencioso Administrativo) y GDPR si aplica.

3. **Plan de migración de datos:** Diseñar estrategia de migración desde FÉNIX, incluyendo mapeo de datos legacy, validación y sincronización inicial.
4. **SLA y acuerdos de servicio:** Definir niveles de servicio (uptime >99.5 %, latencia <5s) y procedimientos de escalamiento ante incidentes.

Para replicación en otros municipios

La arquitectura es replicable en otras ciudades colombianas:

1. **Adaptación normativa local:** Revisar códigos de policía municipales y ajustar tipos de infracciones en Smart Contract según normativa local.
2. **Federación de redes Fabric:** Explorar modelo de red Fabric multi-ciudad donde cada municipio es una organización, permitiendo interoperabilidad nacional.
3. **Compartición de infraestructura:** Evaluar modelo de consorcio donde múltiples municipios pequeños compartan nodos de Ethereum e IPFS para reducir costos.
4. **Estandarización de contratos:** Proponer estándar nacional de Smart Contract para fotomultas, facilitando portabilidad de multas entre jurisdicciones.

Trabajo futuro

Se propone el siguiente roadmap para evolución del proyecto:

Fase 1: Completar Arquitectura Híbrida (3-6 meses)

- Implementar red completa de Hyperledger Fabric con 3 organizaciones (SDM, Policía, Auditoría)
- Desarrollar chaincode completo en Go con Private Data Collections
- Finalizar servicio de sincronización con manejo robusto de errores y reintentos
- Separar IPFS en nodos privado y público con políticas de acceso diferenciadas
- Implementar ServiceFactory para eliminación del patrón Singleton

Fase 2: Despliegue en Servidor Universitario (1-2 meses)

- Migrar Smart Contract de Hardhat local a Sepolia Testnet (o mainnet con Layer 2)
- Configurar nodos IPFS en servidor de la Universidad Distrital
- Desplegar red Hyperledger Fabric en infraestructura universitaria con Docker Swarm
- Configurar CI/CD con GitHub Actions para despliegue automatizado
- Implementar monitoreo con Prometheus y Grafana

Fase 3: Piloto Controlado con SDM (6 meses)

- Firma de convenio con Secretaría Distrital de Movilidad
- Integración real con APIs de SIMIT y RUNT (tramitar credenciales)
- Piloto con 5,000-10,000 multas reales (datos anonimizados si es requerido)
- Evaluación de rendimiento bajo carga real (457,000 comparendos semestrales proyectados)
- Encuestas de satisfacción con agentes de tránsito y ciudadanos
- Medición de reducción efectiva en tasa de impugnación y PQRS

Fase 4: Funcionalidades Avanzadas (6-12 meses)

- **Módulo de Pagos:** Integración con PSE, Nequi, Daviplata para pago de multas desde la plataforma
- **Apelaciones en línea:** Proceso completo de apelación digital con upload de evidencias y resolución transparente
- **Notificaciones automáticas:** Sistema de alertas vía SMS y correo electrónico al registrar multas

- **Dashboard analítico:** Visualizaciones avanzadas para toma de decisiones (zonas con más infracciones, tipos recurrentes, tendencias temporales)
- **API pública:** Endpoints públicos para desarrolladores externos (apps de consulta de multas, integraciones con aseguradoras)

Fase 5: Escalamiento Nacional (12+ meses)

- Extensión a otras ciudades principales (Medellín, Cali, Barranquilla)
- Propuesta de estándar nacional de blockchain para fotomultas al Ministerio de Transporte
- Evaluación de portabilidad normativa y técnica entre jurisdicciones
- Modelo de federación de redes Fabric para interoperabilidad municipal
- Investigación de soluciones Layer 2 (Polygon, Arbitrum) para reducción de costos de gas en Ethereum

Reflexiones finales

Este trabajo demuestra que la tecnología blockchain ha madurado lo suficiente para aplicaciones gubernamentales reales. La arquitectura híbrida propuesta resuelve el dilema fundamental entre privacidad y transparencia que enfrentan las instituciones públicas, proporcionando un modelo técnicamente viable y escalable.

El contexto de Bogotá, con una crisis de confianza documentada (tasa de impugnación del 34.1 %, 155,854 PQRSD semestrales, presunto detrimento patrimonial de \$8,000 millones), presenta una oportunidad única para que blockchain demuestre su valor en casos de uso reales. Si bien el prototipo fue validado en entorno de laboratorio, los fundamentos técnicos son sólidos y la arquitectura es extensible hacia producción.

La transición de modelos de confianza administrativa a confianza criptográfica representa un cambio paradigmático en cómo las instituciones gubernamentales pueden relacionarse

con la ciudadanía. Este trabajo contribuye a ese futuro, proporcionando no solo código funcional sino también un análisis académico riguroso de los desafíos y oportunidades que esta transición implica.

El camino hacia la adopción masiva de blockchain en gobierno es largo, pero trabajos como este, que combinan rigor académico con implementación práctica, acercan ese futuro. La arquitectura híbrida aquí propuesta puede servir de referencia para otros proyectos que requieran balance entre privacidad, transparencia y descentralización.

Anexos

Anexo A: código fuente relevante

Smart Contract FineManagement.sol (Extracto Comentado)

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.28;

import "@openzeppelin/contracts/access/Ownable.sol";

/**
 * @title FineManagement
 * @dev Smart Contract para gestión pública de fotomultas en Ethereum
 * @notice Este contrato almacena metadatos públicos de multas para
 * verificación ciudadana sin comprometer datos sensibles
 */
contract FineManagement is Ownable {
    uint256 private _fineIds;

    // Estados posibles de una multa
    enum FineState {
        PENDING,           // Registrada, pendiente de notificación
        PAID,               // Pagada completamente
        APPEALED,           // En proceso de apelación
        RESOLVED_APPEAL,    // Apelación resuelta
        CANCELLED           // Anulada por error
    }

    // Estructura de datos pública de una multa
```

```
struct Fine {  
    uint256 id;  
    string plateNumber;  
    string evidenceCID;        // CID del hash en IPFS público  
    string location;  
    uint256 timestamp;  
    string infractionType;  
    uint256 cost;  
    string ownerIdentifier;    // Identificador público (no DNI)  
    FineState currentState;  
    address registeredBy;      // Dirección del agente  
    string externalSystemId;   // ID del SIMIT  
}
```

```
// Historial de cambios de estado
```

```
struct FineStatusUpdate {  
    uint256 lastUpdatedTimestamp;  
    FineState oldState;  
    FineState newState;  
    string reason;  
    address updatedBy;  
}
```

```
// Mapings para consultas eficientes
```

```
mapping(uint256 => Fine) public fines;  
mapping(uint256 => FineStatusUpdate[]) public fineStatusHistory;  
mapping(string => uint256[]) public finesByPlate;
```

```
mapping(address => bool) public operators;

// Eventos para auditoría
event FineRegistered(
    uint256 indexed fineId,
    string indexed plateNumber,
    string evidenceCID,
    string ownerIdentifier,
    uint256 cost,
    uint256 timestamp
);

event FineStatusUpdated(
    uint256 indexed fineId,
    FineState indexed oldState,
    FineState indexed newState,
    string reason,
    uint256 timestamp
);

/**
 * @dev Modificador que restringe acceso a operadores autorizados
 */
modifier onlyOperator() {
    require(
        operators[msg.sender] || owner() == msg.sender,
        "Not an operator"
    );
}
```

```
    );  
    _;  
}  
  
/**  
 * @dev Constructor: inicializa el owner como operador  
 */  
constructor() Ownable(msg.sender) {  
    operators[msg.sender] = true;  
}  
  
/**  
 * @notice Registra una nueva multa (solo operadores)  
 * @param _plateNumber Número de placa del vehículo  
 * @param _evidenceCID CID del hash de evidencia en IPFS  
 * @param _location Ubicación de la infracción  
 * @param _infractionType Tipo de infracción (ej. "Exceso velocidad")  
 * @param _cost Valor de la multa en pesos  
 * @param _ownerIdentifier Identificador del propietario  
 * @param _externalSystemId ID opcional del SIMIT  
 * @return newFineId ID asignado a la nueva multa  
 */  
function registerFine(  
    string memory _plateNumber,  
    string memory _evidenceCID,  
    string memory _location,  
    string memory _infractionType,
```

```
    uint256 _cost,
    string memory _ownerIdentifier,
    string memory _externalSystemId
) public onlyOperator returns (uint256) {
    // Validaciones de entrada
    require(bytes(_plateNumber).length > 0,
        "Plate number is required");
    require(bytes(_evidenceCID).length > 0,
        "Evidence CID is required");
    require(_cost > 0, "Cost must be greater than zero");

    // Incrementar contador e ID
    _fineIds += 1;
    uint256 newFineId = _fineIds;

    // Almacenar estructura de multa
    fines[newFineId] = Fine({
        id: newFineId,
        plateNumber: _plateNumber,
        evidenceCID: _evidenceCID,
        location: _location,
        timestamp: block.timestamp,
        infractionType: _infractionType,
        cost: _cost,
        ownerIdentifier: _ownerIdentifier,
        currentState: FineState.PENDING,
        registeredBy: msg.sender,
```

```
        externalSystemId: _externalSystemId
    });

    // Actualizar índices de búsqueda
    finesByPlate[_plateNumber].push(newFineId);

    // Registrar estado inicial en historial
    fineStatusHistory[newFineId].push(FineStatusUpdate({
        lastUpdatedTimestamp: block.timestamp,
        oldState: FineState.PENDING,
        newState: FineState.PENDING,
        reason: "Fine registered",
        updatedBy: msg.sender
    }));

    // Emitir evento
    emit FineRegistered(
        newFineId,
        _plateNumber,
        _evidenceCID,
        _ownerIdentifier,
        _cost,
        block.timestamp
    );

    return newFineId;
}
```

```
/**
 * @notice Actualiza el estado de una multa (solo operadores)
 * @param _fineId ID de la multa a actualizar
 * @param _newState Nuevo estado
 * @param _reason Razón del cambio
 */
function updateFineStatus(
    uint256 _fineId,
    FineState _newState,
    string memory _reason
) public onlyOperator {
    Fine storage fine = fines[_fineId];
    require(fine.id != 0, "Fine does not exist");
    require(fine.currentState != _newState,
        "State is already the same");

    FineState oldState = fine.currentState;
    fine.currentState = _newState;

    // Registrar cambio en historial
    fineStatusHistory[_fineId].push(FineStatusUpdate({
        lastUpdatedTimestamp: block.timestamp,
        oldState: oldState,
        newState: _newState,
        reason: _reason,
        updatedBy: msg.sender
```



```
    });

    emit FineStatusUpdated(
        _fineId,
        oldState,
        _newState,
        _reason,
        block.timestamp
    );
}

/**
 * @notice Consulta pública de detalles de una multa
 * @param _fineId ID de la multa
 * @return fine Estructura completa de la multa
 */
function getFineDetails(uint256 _fineId)
    public view returns (Fine memory)
{
    require(fines[_fineId].id != 0, "Fine does not exist");
    return fines[_fineId];
}

/**
 * @notice Consulta pública por número de placa
 * @param _plateNumber Número de placa
 * @return Array de IDs de multas asociadas
```

```
    */

    function getFinesByPlate(string memory _plateNumber)

        public view returns (uint256[] memory)

    {

        return finesByPlate[_plateNumber];

    }

}
```

Servicio Backend: FineService.ts (Extracto)

```
import { BlockchainService } from './blockchain.service';
import { IPFSService } from './ipfs.service';
import { AptitudeService } from './apitude.service';

/**
 * Servicio principal que orquesta las operaciones de multas
 * Coordina interacción con Ethereum, IPFS y APIs externas
 */
export class FineService {

    private blockchainService: BlockchainService;

    private ipfsService: IPFSService;

    private aptitudeService: AptitudeService;

    constructor() {

        this.blockchainService = BlockchainService.getInstance();

        this.ipfsService = IPFSService.getInstance();

        this.aptitudeService = AptitudeService.getInstance();

    }

}
```

```
/**
 * Registra una nueva multa completa:
 * 1. Upload de evidencia a IPFS
 * 2. Registro en Ethereum
 * 3. (Futuro) Sincronización con Hyperledger
 */
async registerFine(
  file: Express.Multer.File,
  fineData: IFineDetails
): Promise<{ fineId: string; txHash: string; cid: string }> {
  // 1. Subir evidencia a IPFS
  const cid = await this.ipfsService.uploadToIPFS(
    file.buffer,
    file.originalname
  );

  // 2. Consultar datos del RUNT (simulado)
  const vehicleData = await this.apititudeService
    .fetchFineFromAptitude(
      fineData.plateNumber,
      new Date().toISOString()
    );

  // 3. Registrar en Ethereum
  const tx = await this.blockchainService.registerFine({
    ...fineData,
    evidenceCID: cid,
```

```
        ownerIdentifier: vehicleData.ownerName
    });

    return {
        fineId: tx.fineId.toString(),
        txHash: tx.txHash,
        cid: cid
    };
}

/**
 * Verifica integridad cruzada entre blockchain e IPFS
 */
async verifyIntegrity(
    fineId: string
): Promise<{ isValid: boolean; cid: string }> {
    // Obtener CID almacenado en blockchain
    const fine = await this.blockchainService
        .getFineDetails(fineId);

    // Verificar que el archivo existe en IPFS
    const exists = await this.ipfsService
        .verifyFileExists(fine.evidenceCID);

    return {
        isValid: exists,
        cid: fine.evidenceCID
    }
}
```

```
    };  
  }  
}
```

Servicio IPFS: IPFSService.ts (Extracto)

```
import { create, IPFSHTTPClient } from 'ipfs-http-client';  
  
/**  
 * Servicio para interacción con IPFS  
 * Maneja upload y recuperación de evidencias  
 */  
export class IPFSService {  
  private static instance: IPFSService;  
  private ipfs: IPFSHTTPClient;  
  
  private constructor() {  
    this.ipfs = create({  
      url: process.env.IPFS_API_URL || 'http://127.0.0.1:5001'  
    });  
  }  
  
  static getInstance(): IPFSService {  
    if (!IPFSService.instance) {  
      IPFSService.instance = new IPFSService();  
    }  
    return IPFSService.instance;  
  }  
}
```

```
/**
 * Sube un archivo a IPFS y retorna su CID
 * @param fileBuffer Buffer del archivo
 * @param fileName Nombre del archivo
 * @returns Content Identifier (CID) del archivo
 */
async uploadToIPFS(
  fileBuffer: Buffer,
  fileName: string
): Promise<string> {
  const { cid } = await this.ipfs.add(fileBuffer, {
    pin: true, // Mantener el archivo en el nodo
    wrapWithDirectory: false
  });

  console.log('Archivo ${fileName} subido a IPFS: ${cid}');
  return cid.toString();
}

/**
 * Recupera un archivo desde IPFS dado su CID
 * @param cid Content Identifier
 * @returns Buffer del archivo
 */
async getFromIPFS(cid: string): Promise<Uint8Array[]> {
  const chunks: Uint8Array[] = [];
```

```
    for await (const chunk of this.ipfs.cat(cid)) {  
        chunks.push(chunk);  
    }  
  
    return chunks;  
}  
}
```

Anexo B: configuración de infraestructura

Configuración de Hardhat (hardhat.config.cjs)

```
require("@nomicfoundation/hardhat-toolbox");  
require("dotenv").config();  
  
module.exports = {  
    solidity: {  
        version: "0.8.28",  
        settings: {  
            optimizer: {  
                enabled: true,  
                runs: 200 // Optimizado para ejecución frecuente  
            }  
        }  
    },  
    networks: {  
        // Red local para desarrollo  
        localhost: {  
            url: "http://127.0.0.1:8545"  
        },  
    },  
}
```

```
// Red de prueba Sepolia
sepolia: {
  url: process.env.SEPOLIA_RPC_URL,
  accounts: [process.env.PRIVATE_KEY],
  chainId: 11155111
}
},
paths: {
  sources: "./contracts",
  tests: "./test",
  cache: "./cache",
  artifacts: "./artifacts"
}
};
```

Configuración de Docker Compose para Hyperledger Fabric

```
version: '2'
```

```
networks:
```

```
  fotomultas:
```

```
services:
```

```
  # Certificate Authority
```

```
  ca.sdm.fotomultas.com:
```

```
    image: hyperledger/fabric-ca:latest
```

```
    environment:
```

- FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
- FABRIC_CA_SERVER_CA_NAME=ca-sdm


```
volumes:
```

- ./crypto-config/peerOrganizations/sdm.fotomultas.com/ca/:/
etc/hyperledger/fabric-ca-server-config

```
command: sh -c 'fabric-ca-server start -b admin:adminpw'
```

```
networks:
```

- fotomultas

```
# Peer Node - Secretaría de Movilidad
```

```
peer0.sdm.fotomultas.com:
```

```
image: hyperledger/fabric-peer:latest
```

```
environment:
```

- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
- CORE_PEER_ID=peer0.sdm.fotomultas.com
- CORE_PEER_ADDRESS=peer0.sdm.fotomultas.com:7051
- CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.sdm.fotomultas.com:7051
- CORE_PEER_LOCALMSPID=SdmMSP

```
volumes:
```

- /var/run/:/host/var/run/
- ./crypto-config/peerOrganizations/sdm.fotomultas.com/
peers/peer0.sdm.fotomultas.com/msp:/
etc/hyperledger/fabric/msp

```
networks:
```

- fotomultas

```
# Orderer Node
```

```
orderer.fotomultas.com:
```

```
image: hyperledger/fabric-orderer:latest
```

```
environment:
  - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
  - ORDERER_GENERAL_GENESISMETHOD=file
  - ORDERER_GENERAL_GENESISFILE=/
    var/hyperledger/orderer/orderer.genesis.block
  - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
  - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/
    var/hyperledger/orderer/tls/server.crt
  - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/
    var/hyperledger/orderer/tls/server.key
volumes:
  - ./channel-artifacts/genesis.block:/
    var/hyperledger/orderer/orderer.genesis.block
  - ./crypto-config/ordererOrganizations/fotomultas.com/
    orderers/orderer.fotomultas.com/msp:/
    var/hyperledger/orderer/msp
networks:
  - fotomultas
```

Anexo C: manual de instalación

Prerrequisitos

- **Sistema Operativo:** Ubuntu 22.04 LTS o superior
- **Node.js:** Versión 20.18.0 o superior
- **npm:** Versión 10.0.0 o superior
- **Docker:** Versión 24.0+ con Docker Compose 2.20+
- **IPFS Kubo:** Versión 0.34.1 o superior

- **Git:** Para clonar repositorios
- **Go:** Versión 1.21+ (solo para desarrollo de chaincode)

Paso 1: Instalación de Node.js

Instalar nvm (Node Version Manager)

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/
```

```
install.sh | bash
```

Recargar terminal

```
source ~/.bashrc
```

Instalar Node.js 20.18.0

```
nvm install 20.18.0
```

```
nvm use 20.18.0
```

Verificar instalación

```
node --version # Debe mostrar v20.18.0
```

```
npm --version # Debe mostrar 10.0.0 o superior
```

Paso 2: Instalación de IPFS

Descargar IPFS Kubo

```
wget https://dist.ipfs.tech/kubo/v0.34.1/
```

```
kubo_v0.34.1_linux-amd64.tar.gz
```

Extraer archivo

```
tar -xvzf kubo_v0.34.1_linux-amd64.tar.gz
```

Instalar

```
cd kubo

sudo bash install.sh

# Verificar instalación
ipfs --version

# Inicializar IPFS (solo primera vez)
ipfs init

# Iniciar daemon de IPFS
ipfs daemon &
```

Paso 3: Clonar e Instalar el Proyecto

```
# Clonar repositorio
git clone <repository-url>

cd backend

# Instalar dependencias
npm install

# Copiar archivo de variables de entorno
cp env.example .env

# Editar .env con configuración local
nano .env
```

Paso 4: Configurar Variables de Entorno

Editar el archivo `.env` con los siguientes valores:

```
# Servidor
```

```
PORT=3000
```

```
# Blockchain (Hardhat)
```

```
RPC_URL=http://127.0.0.1:8545
```

```
PRIVATE_KEY=0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80
```

```
CONTRACT_ADDRESS= # Se obtiene después del despliegue
```

```
# IPFS
```

```
IPFS_API_URL=http://127.0.0.1:5001
```

```
# APIs Externas (opcional para simulación)
```

```
SIMIT_API_BASE_URL=https://api.simit.com
```

```
SIMIT_API_KEY=your_api_key_here
```

Paso 5: Compilar y Desplegar Smart Contracts

```
# Terminal 1: Compilar contratos
```

```
npm run build:contracts
```

```
# Terminal 2: Iniciar nodo local de Hardhat
```

```
npm run dev:contracts
```

```
# Terminal 3: Desplegar contratos
```

```
npm run deploy
```

```
# Copiar la dirección del contrato desplegado y
```

```
# agregarla al archivo .env como CONTRACT_ADDRESS
```

Paso 6: Iniciar el Backend

```
# Iniciar servidor de desarrollo (con recarga automática)
```

```
npm run dev
```

```
# El servidor estará disponible en http://localhost:3000
```

```
# Documentación Swagger en http://localhost:3000/api-docs
```

Paso 7: Iniciar el Frontend

```
# En directorio del frontend
```

```
cd ../fotomultas-front
```

```
# Instalar dependencias
```

```
npm install
```

```
# Iniciar servidor de desarrollo
```

```
npm run dev
```

```
# El frontend estará disponible en http://localhost:5173
```

Verificación de la Instalación

```
# Verificar conectividad con IPFS
```

```
curl http://localhost:5001/api/v0/version
```

```
# Verificar que Hardhat está corriendo
```

```
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_blockNumber",  
"params":[],"id":1}' http://localhost:8545
```

```
# Verificar backend
```

```
curl http://localhost:3000/api-docs
```

```
# Verificar frontend
```

```
curl http://localhost:5173
```

Anexo D: manual de usuario

Manual para Agentes de Tránsito

1. Iniciar Sesión..

- Acceder a la URL del sistema
- Ingresar credenciales proporcionadas por el administrador
- Seleccionar rol "Agente de Tránsito"

2. Registrar una Multa..

- En el menú principal, seleccionar "Registrar Multa"
- Completar el formulario con:
 - Número de placa del vehículo
 - Tipo de infracción (seleccionar de lista desplegable)
 - Ubicación (GPS automático o manual)
 - Costo de la multa (calculado automáticamente según tipo)
- Cargar evidencia fotográfica (máximo 5MB, formato JPG/PNG)
- Hacer clic en "Registrar Multa"
- Esperar confirmación de blockchain (aprox. 2-5 segundos)
- Anotar el ID de multa generado para referencia

Figura 20

Pantalla de Registro de Multa - Panel del Agente

3. Actualizar Estado de multa..

- Buscar multa por ID o número de placa
- Seleccionar "Actualizar Estado"
- Elegir nuevo estado (Pagada, En Apelación, etc.)
- Ingresar razón del cambio
- Confirmar actualización

Manual para ciudadanos**1. Consultar multas..**

- Acceder a la sección pública (sin autenticación requerida)
- Ingresar número de placa del vehículo
- Hacer clic en "Buscar"
- Revisar lista de multas asociadas

2. Verificar Integridad de evidencia..

- Seleccionar una multa de la lista
- Hacer clic en "Verificar Integridad"
- El sistema compara el hash de la evidencia en blockchain con el archivo en IPFS
- Se muestra resultado: "Evidencia Verificada." "Evidencia Alterada"

Figura 21*Pantalla de Consulta Pública - Panel Ciudadano***3. Presentar apelación..**

- Crear cuenta en el sistema (requiere verificación de identidad)
- Seleccionar multa a apelar
- Completar formulario de apelación con argumentos
- Cargar evidencias de respaldo (opcional)
- Enviar apelación
- Esperar notificación de resolución (máximo 30 días hábiles)

Anexo E: glosario de términosTabla 15. *Glosario de Términos Técnicos*

Término	Definición
ABI (Application Binary Interface)	Interfaz que define cómo llamar funciones de un Smart Contract desde aplicaciones externas. Contiene nombres de funciones, parámetros y tipos de retorno.
Blockchain	Tecnología de registro distribuido que almacena datos en bloques encadenados mediante hashes criptográficos, garantizando inmutabilidad.
CA (Certificate Authority)	Entidad que emite y gestiona certificados digitales en una red Hyperledger Fabric, controlando identidades y permisos.

Cuadro 15 – continuación de la página anterior

Término	Definición
Chaincode	Smart Contract en el contexto de Hyperledger Fabric, generalmente escrito en Go, que define la lógica de negocio.
CID (Content Identifier)	Hash único que identifica un archivo en IPFS. Se genera mediante criptografía del contenido del archivo.
Consenso	Mecanismo mediante el cual los nodos de una blockchain acuerdan la validez de las transacciones. Ejemplos: PBFT, PoS, PoW.
DLT (Distributed Ledger Technology)	Tecnología de libro mayor distribuido que mantiene registros sincronizados entre múltiples nodos sin autoridad central.
Ethers.js	Biblioteca JavaScript para interactuar con la blockchain de Ethereum, permitiendo leer datos y enviar transacciones.
Gas	Unidad de medida del costo computacional en Ethereum. Cada operación consume gas que se paga en Ether.
Hardhat	Framework de desarrollo para Ethereum que facilita compilación, testing y despliegue de Smart Contracts.
Hash Criptográfico	Función matemática que convierte datos de cualquier tamaño en una cadena de longitud fija. Ejemplos: SHA-256, Keccak-256.
Hyperledger Fabric	Plataforma de blockchain permissionada empresarial, parte del proyecto Hyperledger de Linux Foundation.
Inmutabilidad	Propiedad de blockchain que garantiza que datos una vez escritos no pueden ser alterados sin dejar evidencia.

Cuadro 15 – continuación de la página anterior

Término	Definición
IPFS (InterPlanetary File System)	Sistema de archivos peer-to-peer distribuido que usa direccionamiento por contenido mediante CIDs.
Ledger	Libro mayor que registra todas las transacciones en una blockchain. Es distribuido y sincronizado entre nodos.
Nodo (Node)	Computadora que participa en una red blockchain, manteniendo una copia del ledger y validando transacciones.
OpenZeppelin	Librería de Smart Contracts auditados y seguros para Ethereum, proporciona implementaciones estándar de tokens, control de acceso, etc.
Orderer	Nodo en Hyperledger Fabric que ordena transacciones y las agrupa en bloques para distribuir a los peers.
PBFT (Practical Byzantine Fault Tolerance)	Algoritmo de consenso tolerante a fallas bizantinas usado en Hyperledger Fabric, eficiente para redes permissionadas.
Peer	Nodo en Hyperledger Fabric que mantiene una copia del ledger y ejecuta chaincode.
Pinning	En IPFS, mantener un archivo almacenado permanentemente en un nodo para garantizar su disponibilidad.
PoS (Proof of Stake)	Mecanismo de consenso donde validadores son seleccionados según la cantidad de criptomoneda que poseen.
PoW (Proof of Work)	Mecanismo de consenso que requiere resolver acertijos criptográficos complejos para validar bloques.

Cuadro 15 – continuación de la página anterior

Término	Definición
Private Data Collections	Funcionalidad de Hyperledger Fabric para almacenar datos privados que solo ciertos nodos pueden acceder.
Smart Contract	Programa autoejecutante almacenado en blockchain que ejecuta lógica de negocio cuando se cumplen condiciones.
Solidity	Lenguaje de programación orientado a objetos para escribir Smart Contracts en Ethereum.
Testnet	Red de prueba de blockchain que imita el funcionamiento de la red principal pero sin valor real. Ejemplo: Sepolia.
Transaction Hash	Identificador único de una transacción en blockchain, generado mediante hash criptográfico de su contenido.
TypeScript	Superset de JavaScript con tipado estático, usado para desarrollo backend del proyecto.
Wallet	Software que almacena claves privadas y permite firmar transacciones en blockchain.