Manual Técnico
Sistema DICRI Evidencias

Ministerio Público de Guatemala
Dirección de Investigación Criminalística (DICRI)

Autor: *Guillermo Enrique Gamboa Rodríguez*
Rol: *Desarrollador – Prueba Técnica*
Fecha: *Noviembre 2025*

Índice

1. Introducción

El presente manual técnico describe la arquitectura, funcionamiento interno, diseño de datos, componentes y proceso de despliegue del sistema DICRI Evidencias.

Este documento está dirigido a desarrolladores, personal técnico y personal de soporte del Ministerio Público.

El objetivo principal es proporcionar una guía clara y estructurada sobre cómo está construido el sistema y cómo mantenerlo.

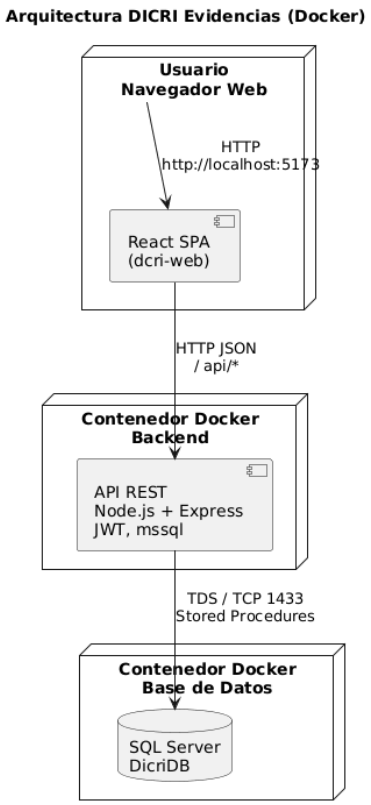
2. Arquitectura del Sistema

El sistema se compone de tres módulos principales, todos desplegados mediante Docker Compose:

- Frontend (React + Vite): Interfaz web institucional donde técnicos y coordinadores gestionan expedientes.
- API Backend (Node.js + Express): Expone endpoints REST, maneja autenticación JWT, roles y consulta estructurada a la base de datos.
- Base de Datos (SQL Server en Docker): Hostea el modelo relacional que soporta expedientes, indicios y trazabilidad del historial.

El flujo general es:

- El usuario interactúa desde el navegador con el frontend.
- El frontend comunica peticiones a la API por HTTP.
- La API ejecuta procedimientos almacenados (SP) en SQL Server.



3. Frontend (React + Vite)

El frontend es una SPA construida con React y Vite. Administra autenticación por JWT, enrutamiento y todas las vistas del sistema:

- Pantalla de Login institucional estilo MP.
- Dashboard de expedientes.
- Registro de expediente.
- Registro y visualización de indicios.
- Revisión de coordinador (aprobación / rechazo).

Estructura principal:

src/

pages/

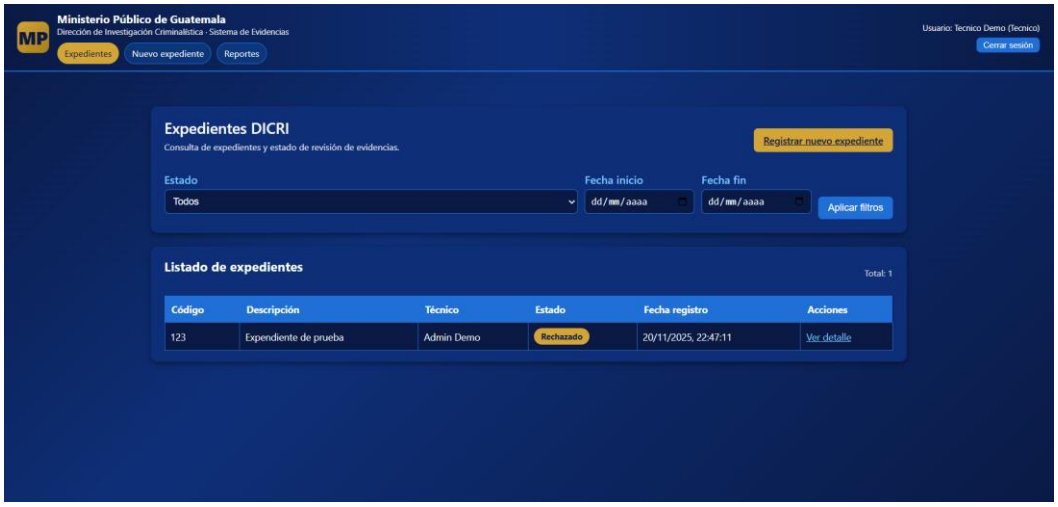
components/

api/

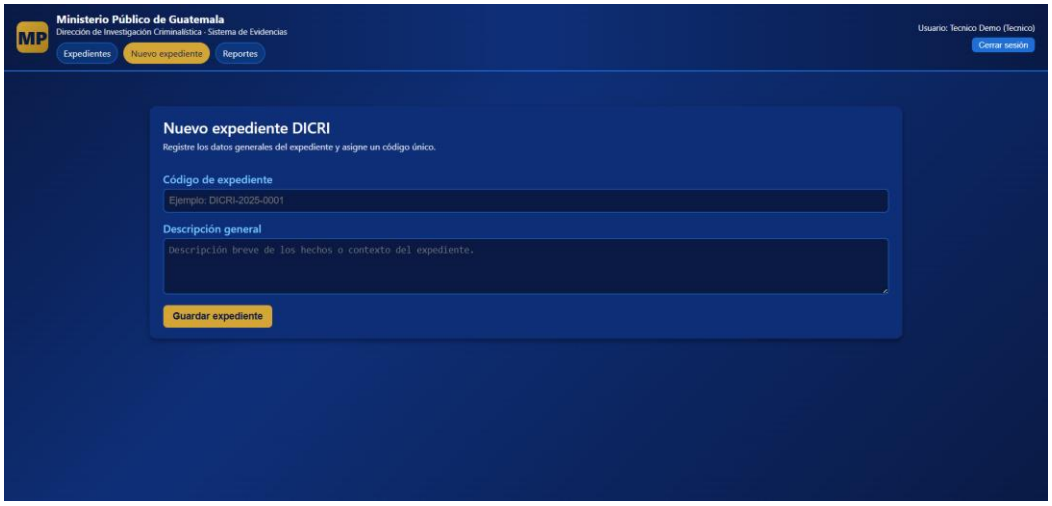
context/

styles.css

Pantalla de inicio de sesión institucional (MP). Implementada en React, con autenticación JWT contra el backend.



Vista principal donde el usuario técnico o coordinador consulta expedientes con filtros por estado y fecha.



Formulario para registrar un expediente nuevo. Incluye validación de código y descripción.

4. Backend API (Node.js + Express)

El backend implementa toda la lógica de negocio y manejo de roles.

Principales componentes:

- authRoutes: login, generación y validación de JWT.
- expedienteRoutes: registrar, listar, obtener y cambiar estado de expedientes.
- indicioRoutes: agregar y listar indicios.
- historialRoutes: obtener historial de estados.
- Middlewares: authenticate y authorizeRoles.

Ejemplo de ruta:

/api/expedientes/:id/rechazar

Ejemplo de middleware:

- authenticate: valida JWT

- authorizeRoles: restringe acceso según rol

Este fragmento muestra la definición de rutas del módulo de expedientes, donde se exponen endpoints REST protegidos por autenticación JWT y autorizaciones basadas en rol.

Dicri Evidencias > backend > src > routes > JS expedienteRoutes.js > ...

```js
import { Router } from "express";
import {
  crearExpedienteController,
  listarExpedientesController,
  aprobarExpedienteController,
  rechazarExpedienteController
} from "../controllers/expedienteController.js";
import { authenticate, authorizeRoles } from "../middleware/authMiddleware.js";

const router = Router();

router.use(authenticate);

router.post("/", authorizeRoles("Tecnico", "Administrador"), crearExpedienteController);
router.get("/", authorizeRoles("Tecnico", "Coordinador", "Administrador"), listarExpedientesController);
router.put("/:id/aprobar", authorizeRoles("Coordinador", "Administrador"), aprobarExpedienteController);
router.put("/:id/rechazar", authorizeRoles("Coordinador", "Administrador"), rechazarExpedienteController);

export default router;

```

Middleware responsable de validar el token JWT enviado por el frontend, decodificarlo y adjuntar la información del usuario autenticado a la solicitud.

Dicri Evidencias > backend > src > middleware > JS authMiddleware.js > ...

```js
import jwt from "jsonwebtoken";
import { jwtConfig } from "../config/jwtConfig.js";

export const authenticate = (req, res, next) => {
  const header = req.headers.authorization;
  if (!header) return res.status(401).json({ message: "No autorizado" });

  const [, token] = header.split(" ");
  try {
    const payload = jwt.verify(token, jwtConfig.secret);
    req.user = payload;
    next();
  } catch (err) {
    return res.status(401).json({ message: "Token inválido" });
  }
};

export const authorizeRoles = (...rolesPermitidos) => {
  return (req, res, next) => {
    if (!req.user || !rolesPermitidos.includes(req.user.role)) {
      return res.status(403).json({ message: "Acceso denegado" });
    }
    next();
  };
};

```

5. Base de Datos – Modelo Relacional

La base de datos DICRI posee las siguientes tablas principales:

- Usuarios
- Roles
- Expedientes
- Indicios

- EstadosExpediente
- HistorialEstados

Todas las operaciones críticas se realizan por Stored Procedures (SP), lo cual asegura seguridad y consistencia.

Ejemplo de tablas:

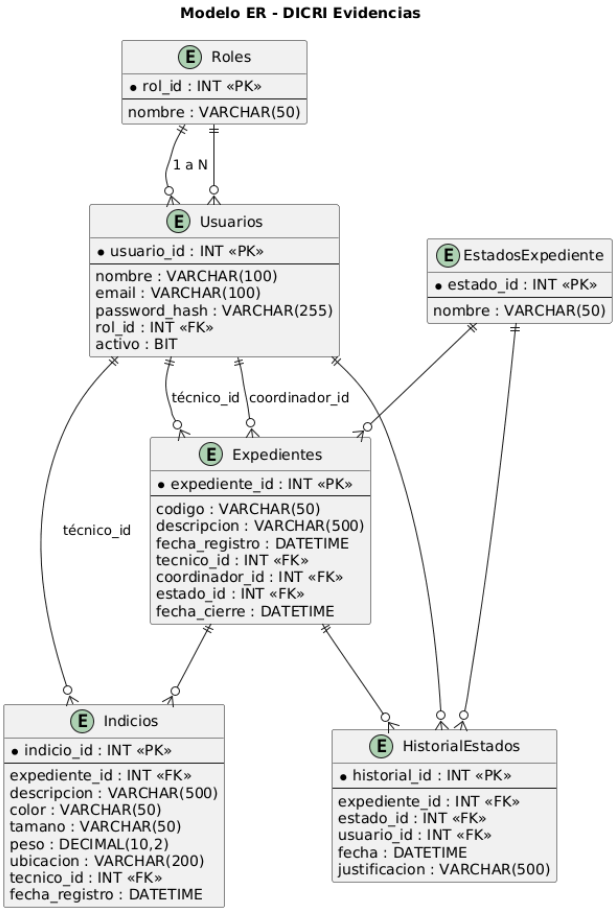
- Expedientes: código, descripción, estado, técnico, fechas.

- Indicios: descripción, ubicación, peso, técnico.

- HistorialEstados: estado, usuario, fecha, justificación.

El modelo entidad–relación define las tablas principales del sistema: Usuarios, Roles, Expedientes, Indicios, EstadosExpediente y HistorialEstados.

Cada expediente tiene un técnico responsable y puede contener múltiples indicios. Los cambios de estado de cada expediente se registran en HistorialEstados junto con el usuario que realizó la acción y la justificación correspondiente, lo que permite mantener trazabilidad completa del ciclo de vida del expediente.



La tabla Expedientes almacena la información principal del caso, incluyendo código, descripción, técnico responsable, coordinador, estado actual y fechas de registro y cierre.

```sql
CREATE TABLE Expedientes (
    expediente_id INT IDENTITY(1,1) PRIMARY KEY,
    codigo VARCHAR(50) NOT NULL UNIQUE,
    descripcion VARCHAR(500) NULL,
    fecha_registro DATETIME NOT NULL DEFAULT GETDATE(),
    tecnico_id INT NOT NULL,
    coordinador_id INT NULL,
    estado_id INT NOT NULL,
    fecha_cierre DATETIME NULL,
    FOREIGN KEY (tecnico_id) REFERENCES Usuarios(usuario_id),
    FOREIGN KEY (coordinador_id) REFERENCES Usuarios(usuario_id),
    FOREIGN KEY (estado_id) REFERENCES EstadosExpediente(estado_id)
);
```

El procedimiento spExpedientes_Insert encapsula la lógica de creación de un expediente. Además de insertar el registro, inicializa el historial de estados con el estado "Registrado".

```sql
CREATE OR ALTER PROCEDURE spExpedientes_Insert
  @codigo VARCHAR(50),
  @descripcion VARCHAR(500),
  @tecnico_id INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @estadoRegistrado INT;
    SELECT @estadoRegistrado = estado_id FROM EstadosExpediente WHERE nombre = 'Registrado';

    INSERT INTO Expedientes (codigo, descripcion, tecnico_id, estado_id)
    VALUES (@codigo, @descripcion, @tecnico_id, @estadoRegistrado);

    DECLARE @nuevoId INT = SCOPE_IDENTITY();

    INSERT INTO HistorialEstados (expediente_id, estado_id, usuario_id, justificacion)
    VALUES (@nuevoId, @estadoRegistrado, @tecnico_id, 'Creación de expediente');

    SELECT @nuevoId AS expediente_id;
END;
GO
```

6. Historial de Estados y Rechazos

Cada cambio de estado se almacena en la tabla HistorialEstados.

Incluye:

- qué usuario realizó el cambio
- fecha del cambio

- estado asignado (Registrado, Revisión, Rechazado, Aprobado)
- justificación cuando aplica

Stored Procedure responsable:

spHistorial_Expediente

El procedimiento spHistorial_Expediente devuelve el historial completo de cambios de estado de un expediente, incluyendo quién realizó el cambio, la fecha y la justificación. Se utiliza en el frontend para mostrar al técnico el motivo del rechazo.

```sql
CREATE OR ALTER PROCEDURE spHistorial_Expediente
  @expediente_id INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        h.historial_id,
        h.fecha,
        est.nombre AS estado,
        u.nombre AS usuario,
        h.justificacion
    FROM HistorialEstados h
    INNER JOIN EstadosExpediente est ON h.estado_id = est.estado_id
    INNER JOIN Usuarios u ON h.usuario_id = u.usuario_id
    WHERE h.expediente_id = @expediente_id
    ORDER BY h.fecha DESC;
END;
GO
```

7. Docker y Orquestación

El sistema se ejecuta en tres contenedores:

- dicri-web (React)
- dicri-api (Node.js)
- dicri-db (SQL Server 2022)

docker-compose.yml define toda la infraestructura:

• networks

• environment variables

• puertos expuestos

• persistencia de datos SQL

Comandos principales:

docker-compose up --build

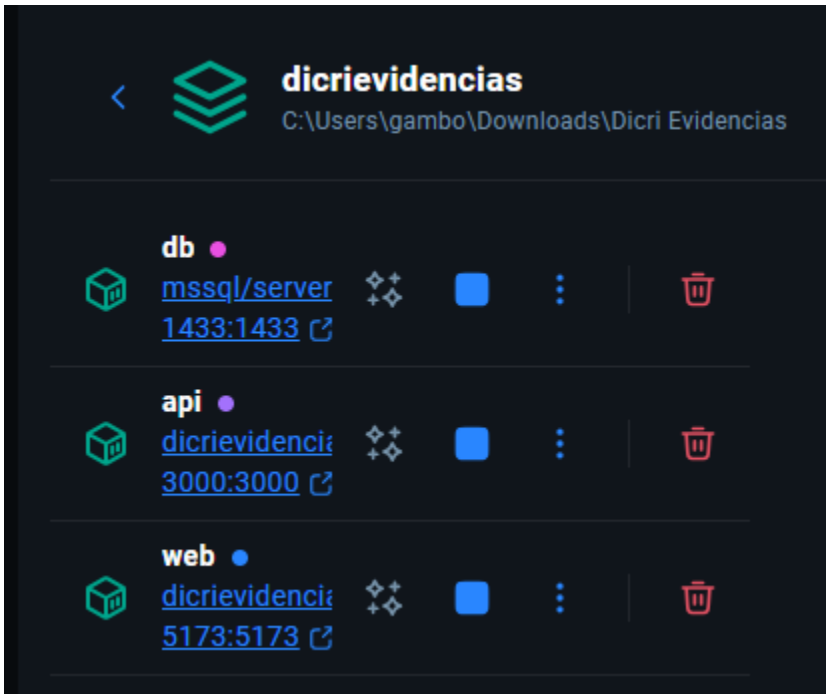
docker-compose down

docker system prune -a

Dicri Evidencias > docker-compose.yml

```yaml
version: "3.9"

services:
  db:
    image: mcr.microsoft.com/mssql/server:2022-latest
    container_name: dicri-db
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=YourStrong!Passw0rd
      - MSSQL_PID=Express
    ports:
      - "1433:1433"
    volumes:
      - mssqldata:/var/opt/mssql

  api:
    build: ./backend
    container_name: dicri-api
    environment:
      - DB_USER=sa
      - DB_PASSWORD=YourStrong!Passw0rd
      - DB_SERVER=db
      - DB_NAME=DicriDB
      - JWT_SECRET=supersecret
    ports:
      - "3000:3000"
    depends_on:
      - db

  web:
    build: ./frontend
    container_name: dicri-web
    ports:
      - "5173:5173"
    depends_on:
      - api

volumes:
  mssqldata:

```

Archivo principal de orquestación que define los servicios dockerizados: frontend (React), backend (Node.js/Express) y base de datos (SQL Server). Controla puertos, variables de entorno, redes y dependencias entre contenedores.



Contenedores activos del sistema DICRI Evidencias en Docker Desktop. Se observan los servicios dicri-db (Base de datos SQL Server), dicri-api (Backend Node.js) y dicri-web (Frontend React) ejecutándose de forma orquestada.

8. Manual de Despliegue

Esta sección describe el proceso completo para desplegar el sistema DICRI Evidencias utilizando Docker y SQL Server, asegurando que todos los componentes (base de datos, backend y frontend) queden funcionando correctamente.

8.1 Requisitos previos

Antes del despliegue, el equipo técnico debe contar con:

- Docker Desktop instalado (Windows o Linux)
- SQL Server Management Studio (SSMS) para ejecutar los scripts
- ZIP o repositorio del proyecto completo

8.2 Pasos para desplegar el sistema

1. Obtener el proyecto
    a. Clonar desde repositorio, git clone https://github.com/G-Gamboa/SistemaDicri

2. Abrir la terminal en la carpeta raíz del proyecto
3. Construir y levantar los contenedores
    a. Ejecutar: docker-compose up –build

8.3 Configuración de la base de datos

1. Abrir SQL Server Management Studio (SSMS)
2. Conectarse al servidor SQL del contenedor
   Usar los siguientes datos:
    - Servidor: localhost,1433
    - Usuario: sa
    - Contraseña: P@assw0rd123
3. Ejecutar los scripts de base de datos
   En SSMS:
    - Abrir db/schema.sql → ejecutar
    - Abrir db/procedures.sql → ejecutar

8.4 Acceso al sistema

Abrir el navegador, ingresando a http://localhost:5173

Allí aparecerá la pantalla de Login institucional.

8.5 Credenciales de prueba

El sistema incluye tres usuarios precargados para ver cada rol:

Técnico:

tecnico@dicri.local / 123456

Coordinador:

coordinador@dicri.local / 123456

Administrador:

admin@dicri.local / 123456

8.6 Flujo sugerido de validación

Después del despliegue, se recomienda verificar:

- Crear expediente (como técnico)
- Agregar indicios
- Iniciar sesión como coordinador
- Aprobar o rechazar expediente
- Ver historial de estado
- Consultar reportes

8.7 Apagado del sistema

Para apagar los contenedores: docker-compose down

Para limpiar imágenes y contenedores no usados: docker system prune -a