21-11-2025

Manual Técnico: Sistema DICRI

Por: Jose Víctor Castellanos Pérez

Prueba técnica

Contenido

INTRODUCCIÓN

1.1. Descripción del proyecto

El presente proyecto consiste en un sistema para la Dirección de Investigación Criminalística (DICRI), en el cual los técnicos pueden gestionar expedientes y asignar indicios a los mismos. Así mismo, los coordinadores tienen la facilidad de ver los detalles de los expedientes a los que fueron asignados, para aprobar o rechazarlos.

1.2. Objetivo del sistema

El objetivo general es facilitar la administración de expedientes por medio de una interfaz intuitiva y eficiente, para mejorar los procesos dentro de DICRI. De igual forma, se busca reducir el tiempo de espera y automatizar el apartado de reportes.

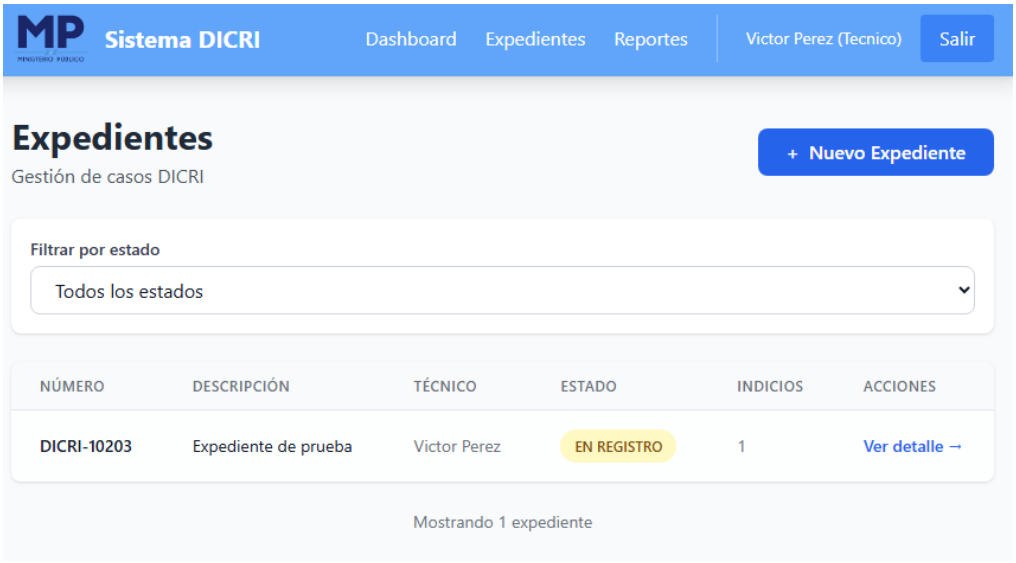
1.3. Alcance del sistema

1.3.1. Alcance funcional

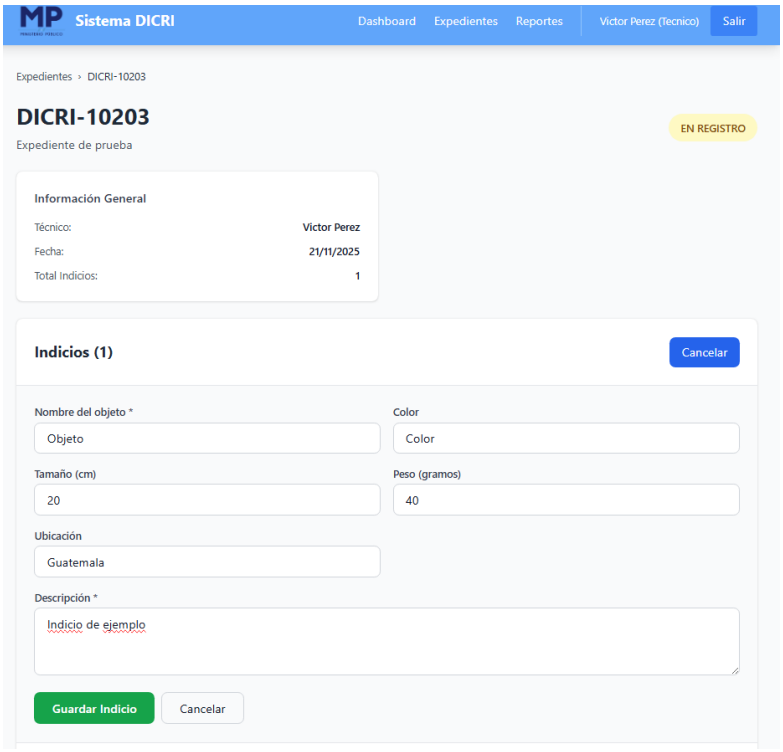
El sistema cuenta con diferentes módulos habilitados según los permisos con los que cuenta cada usuario, entre estos se encuentran:

- **Módulo de autenticación y control de acceso**: Un sistema de inicio de sesión seguro en el cual el usuario ingresa sus credenciales para acceder al sistema. Dependiendo de su rol, obtiene permisos diferentes. Los tres roles que se manejan son: Administrador, coordinador y técnico. El sistema de autenticación se basa en JWT y tiene un tiempo de expiración configurable

- **Módulo de expedientes**: En este apartado, los técnicos tienen la opción de gestionar los expedientes. Permite añadir nuevos y ver los detalles y estado de estos. El sistema maneja cuatro estados para los expedientes: EN_REGISTRO, EN_REVISION, APROBADO y RECHAZADO.

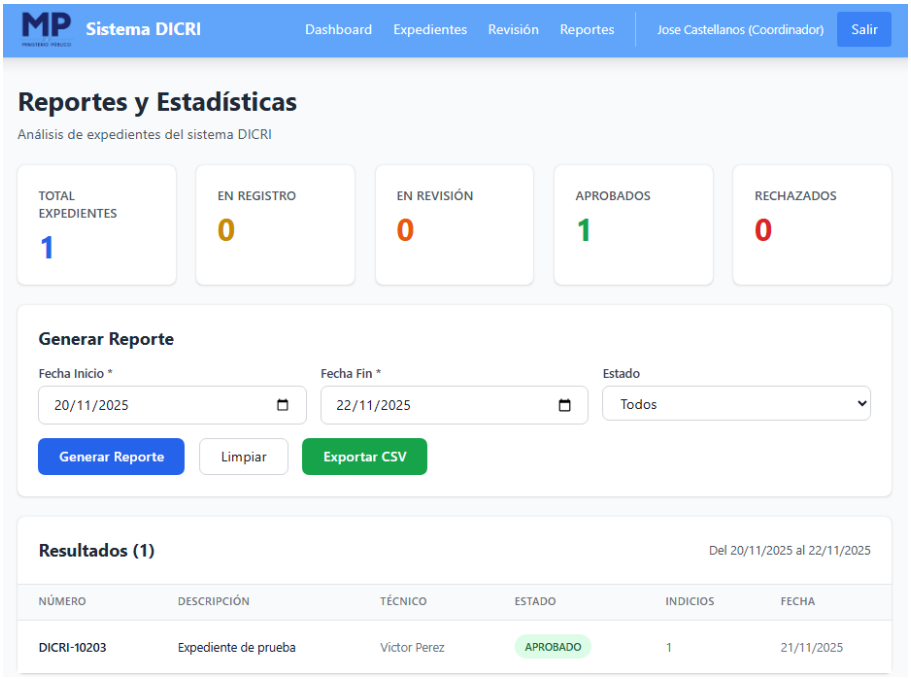


- **Módulo de gestión de indicios**: En este módulo se pueden crear indicios y asignarlos a un expediente en específico. Se cuenta con parámetros de alta importancia como nombre del objeto, descripción, color, tamaño, peso, etc.



- **Módulo de revisión y aprobación**: Aquí los coordinadores pueden ver todos los expedientes que se les han sido asignados para revisión, tienen la opción de ver los detalles e indicios de cada uno, y aprobarlos o rechazarlos según su consideración. En caso de ser rechazados, deben colocar una justificación del rechazo, y esta aparecerá junto al estado del expediente.

- **Módulo de reportes y estadísticas**: Se cuenta con un apartado de reportes en el cual se puede filtrar el historial de expedientes por fecha o por estado. Así mismo, se pueden importar los resultados como CSV para su posterior análisis.



ARQUITECTURA DEL SISTEMA

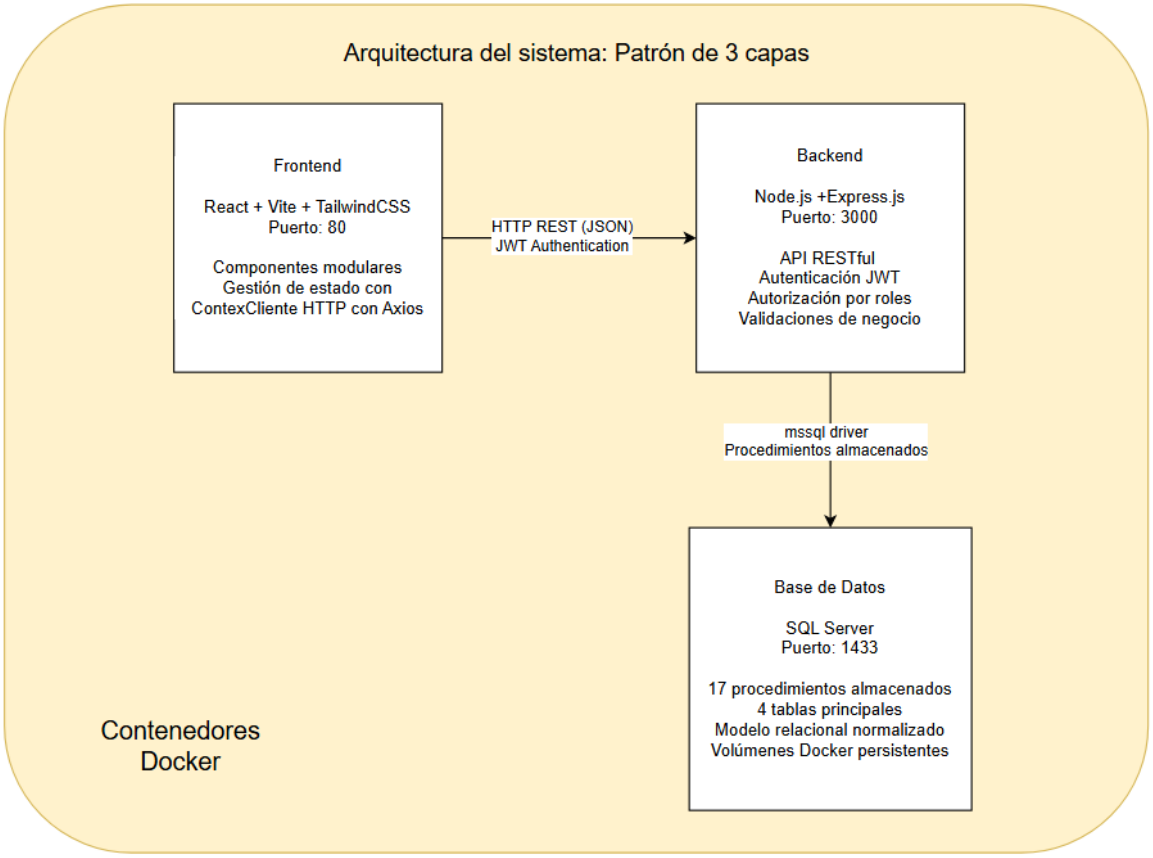
2.1. Patrón de arquitectura

La arquitectura del sistema consiste en un patrón de tres capas principales. Esto permite facilita considerablemente el mantenimiento a largo plazo y ayuda a distribuir responsabilidades entre diferentes áreas. Las tres capas consisten en:

- **Frontend**: Se implementó con React 18.3.1, esta capa es la encargada de todo el apartado visual del sistema y de toda la interacción con el usuario. Se crearon componentes para facilitar la reutilización y para manejar modularidad en el sistema. Además, se utilizó TailwindCSS para el apartado visual, lo que proporciona una experiencia de usuario agradable con diseño estandarizado y cómodo a la vista.
- **Backend**: Se construyó haciendo uso de Node.js con Express.js. En esta capa se almacena toda la lógica del del sistema. Se implementó una API RESTful que sigue buenas prácticas de diseño y se utilizan métodos HTTP apropiados para cada una de las diferentes operaciones.
- **Capa de datos**: Para este apartado se utilizó SQL Server, y aquí se gestiona todo lo relacionado a la persistencia de los datos. Se hizo uso de procedimientos almacenados para todas las operaciones, lo que mejora la seguridad del sistema y previene ataques como

inyecciones. Se diseño un modelo relacional normalizado, lo que garantiza la integridad de los datos

2.2. Diagrama de arquitectura del sistema



TECNOLOGÍAS UTILIZADAS

Se seleccionaron estas herramientas considerando diversos factores como la capacidad de escalamiento futuro, soporte de la comunidad y madurez. Entre estas se encuentran:

3.1. Stack Frontend

- **React:** Se tomó como base para construir las pantallas del sistema de evidencias. Permite organizar la interfaz en componentes reutilizables (tablas, formularios, layouts) y mantener el código de las vistas más ordenado.
- **Vite:** Se utilizó como herramienta de arranque del proyecto y entorno de desarrollo. Facilita trabajar con recarga rápida de cambios y una configuración sencilla, algo útil cuando se están probando varios flujos de la aplicación.

- **TailwindCSS:** Se empleó para el diseño visual. En lugar de mantener hojas de estilo extensas, se añadieron clases utilitarias directamente en los componentes, lo que hizo más fácil ajustar espaciados, tamaños y colores hasta lograr una interfaz limpia.
- **React Router:** Se usó para manejar la navegación interna entre módulos (dashboard, expedientes, revisión, reportes). Gracias a esto se cambian de vista las secciones sin recargar toda la página.
- **Axios:** Se eligió como cliente HTTP para comunicarse con la API. Se configuró una instancia con la URL base del backend y el envío del token de autenticación, de modo que las llamadas desde los componentes quedan centralizadas en los servicios.

3.2. Stack Backend

- **Node.js:** En este proyecto se tomó como entorno del lado del servidor; toda la lógica del sistema de evidencias corre ahí y se mantiene un solo lenguaje (JavaScript) en todo el desarrollo.
- **Express:** Sobre Node.js se montó Express para armar la API REST. Con él se agruparon las rutas por módulo (login, expedientes, indicios, reportes) y se encadenaron los middlewares que validan datos y autorizan a cada rol.
- **Mssql:** Es la librería que se usa para conectarse a SQL Server. Desde los controladores solo se invocan los procedimientos almacenados (sp_CrearExpediente, sp_CrearIndicio, etc.), sin escribir consultas SQL directas en el código.
- **Jsonwebtoken:** Aquí se encarga de crear y revisar los tokens JWT que se entregan al iniciar sesión; ese token acompaña las peticiones a los endpoints que requieren usuario autenticado.
- **Bcryptjs:** Se emplea para calcular el hash de las contraseñas antes de guardarlas en la tabla Usuario, de modo que nunca se almacena el valor original que escribió la persona.
- **Cors:** Middleware activado para que la aplicación React, que corre en otro origen, pueda llamar a la API sin que el navegador bloquee las solicitudes por políticas de seguridad.
- **Dotenv:** Carga desde un archivo .env la configuración sensible del sistema (cadena de conexión, claves de JWT, puertos), lo que evita dejar esos datos escritos directamente en el repositorio.

MODELO DE LA BASE DE DATOS

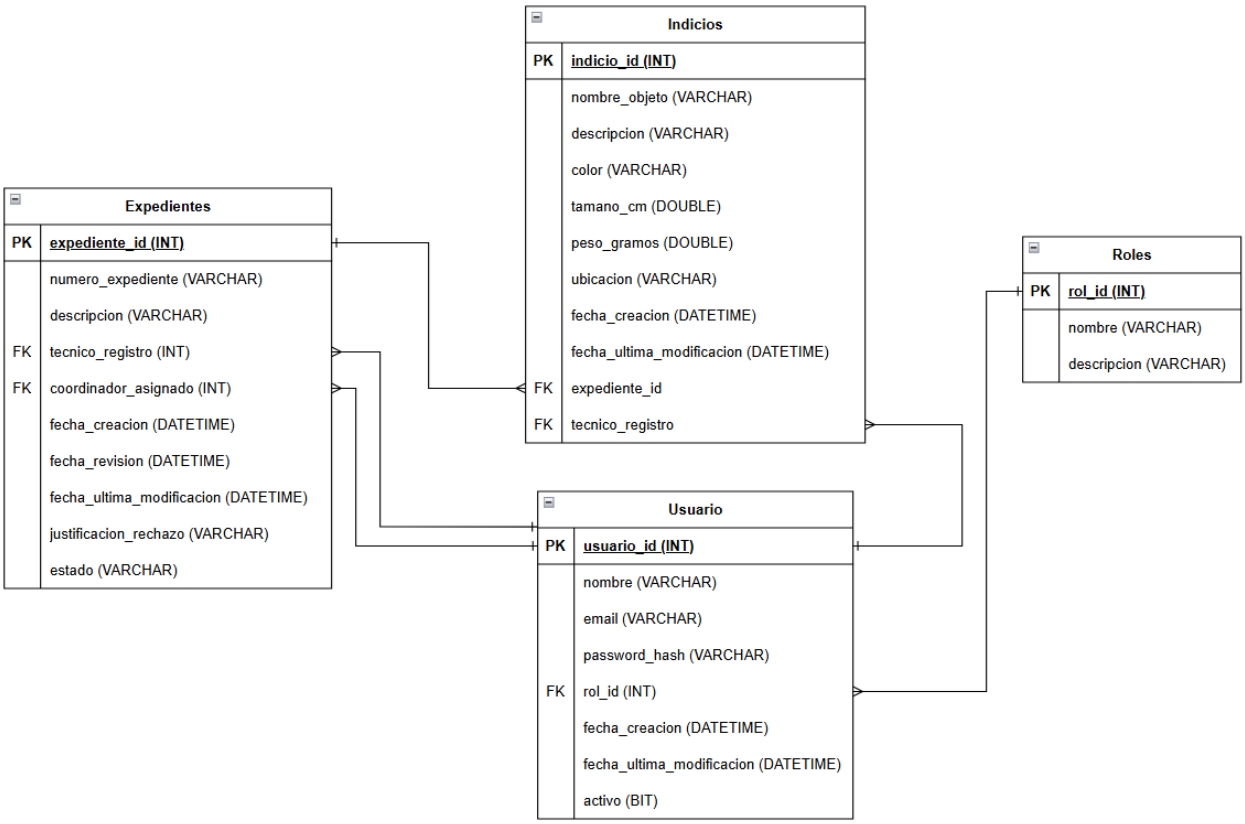
4.1. Diseño del Modelo Relacional

El modelo relacional de la base de datos se diseñó con el objetivo de cubrir todas las necesidades del proyecto sin necesidad de complicar nada de manera innecesaria. Se cuentan con cuatro tablas principales, que almacenan información indispensable sobre Usuarios, Roles, Indicios y Expedientes.

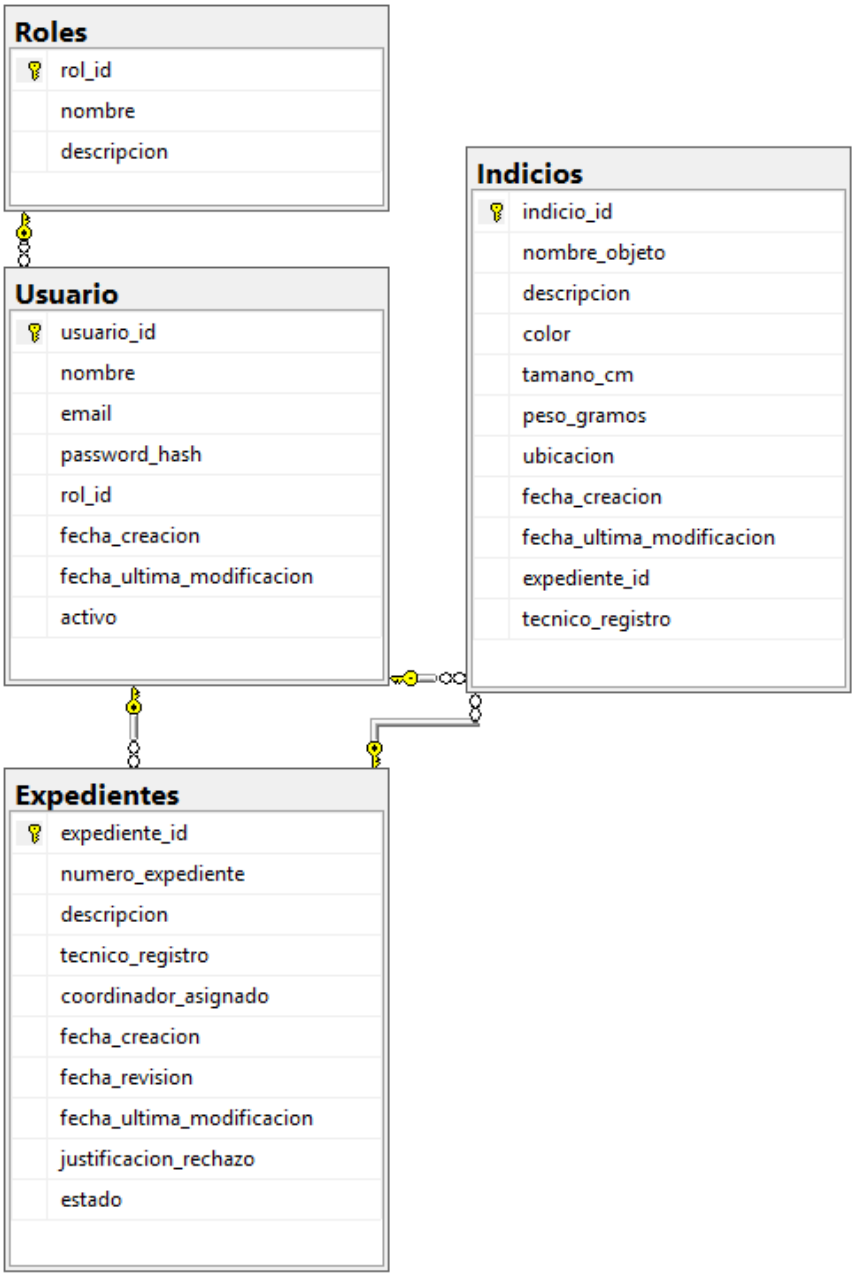
4.2. Descripción detallada de tablas

- **Usuario**: En esta tabla se encuentra la información de cada uno de los empleados con acceso al sistema, aquí se almacena el nombre, correo, contraseña, fecha de creación, fecha de última modificación, el rol (que viene de su respectiva tabla) y el estado de la cuenta, ya sea activo o inactivo.
- **Roles**: Esta es una tabla pequeña pero importante, pues define el puesto del usuario, y, por ende, los permisos que posee. Se cuentan con tres roles: El administrador tiene acceso a todo el sistema, un técnico tiene la capacidad de crear expedientes y asignar indicios, y un coordinador recibe los expedientes y tiene la opción de aprobarlos o rechazarlos.
- **Expedientes**: Esta tabla almacena información como número de expediente, descripción, técnico registrado, coordinador asignado, fecha de creación, fecha de revisión, fecha de última modificación, justificación de rechazo en caso de que no haya sido aprobado y estado del expediente.
- **Indicios**: Esta tabla almacena cualquier tipo de evidencia recolectada en la escena, estos van asignados directamente a los expedientes. Contiene información como nombre, descripción, color, tamaño, peso, ubicación, fecha de última modificación, expediente al que fue asignado y técnico que lo registró

4.3. Diseño inicial del modelo relacional



4.4. Modelo relacional implementado



PROCEDIMIENTOS ALMACENADOS

5.1. Resumen de procedimientos almacenados por categoría

Categoría	Cantidad	Procedimiento Almacenado
Autenticación	2	sp_CrearUsuario, sp_ObtenerUsuarios
Expedientes	7	sp_CrearExpediente, sp_ObtenerExpedientes, sp_ObtenerExpedientePorId, sp_ActualizarExpediente, sp_EnviarExpedienteRevision, sp_AprobarExpediente, sp_RechazarExpediente
Indicios	4	sp_CrearIndicio, sp_ObtenerIndiciosPorExpediente, sp_ActualizarIndicio, sp_EliminarIndicio
Reportes	4	sp_ObtenerReportePorFechas, sp_ObtenerEstadisticas, sp_ObtenerExpedientesPorTecnico, sp_ObtenerExpedientesParaRevision

5.2. Ejemplos de procedimientos almacenados clave

5.2.1. sp_crearExpediente

Aspecto	Detalle
Propósito	Registra un nuevo expediente en el sistema con validaciones de negocio
Parámetros de Entrada	@numero_expediente VARCHAR(50), @descripcion VARCHAR(500), @tecnico_registro INT
Valor de Retorno	expediente_id del nuevo registro creado

Validaciones	Verifica que el número de expediente sea único en el sistema
Manejo de Transacciones	Utiliza BEGIN TRANSACTION con ROLLBACK automático en caso de error
Estado Inicial	El expediente se crea automáticamente en estado "EN_REGISTRO"

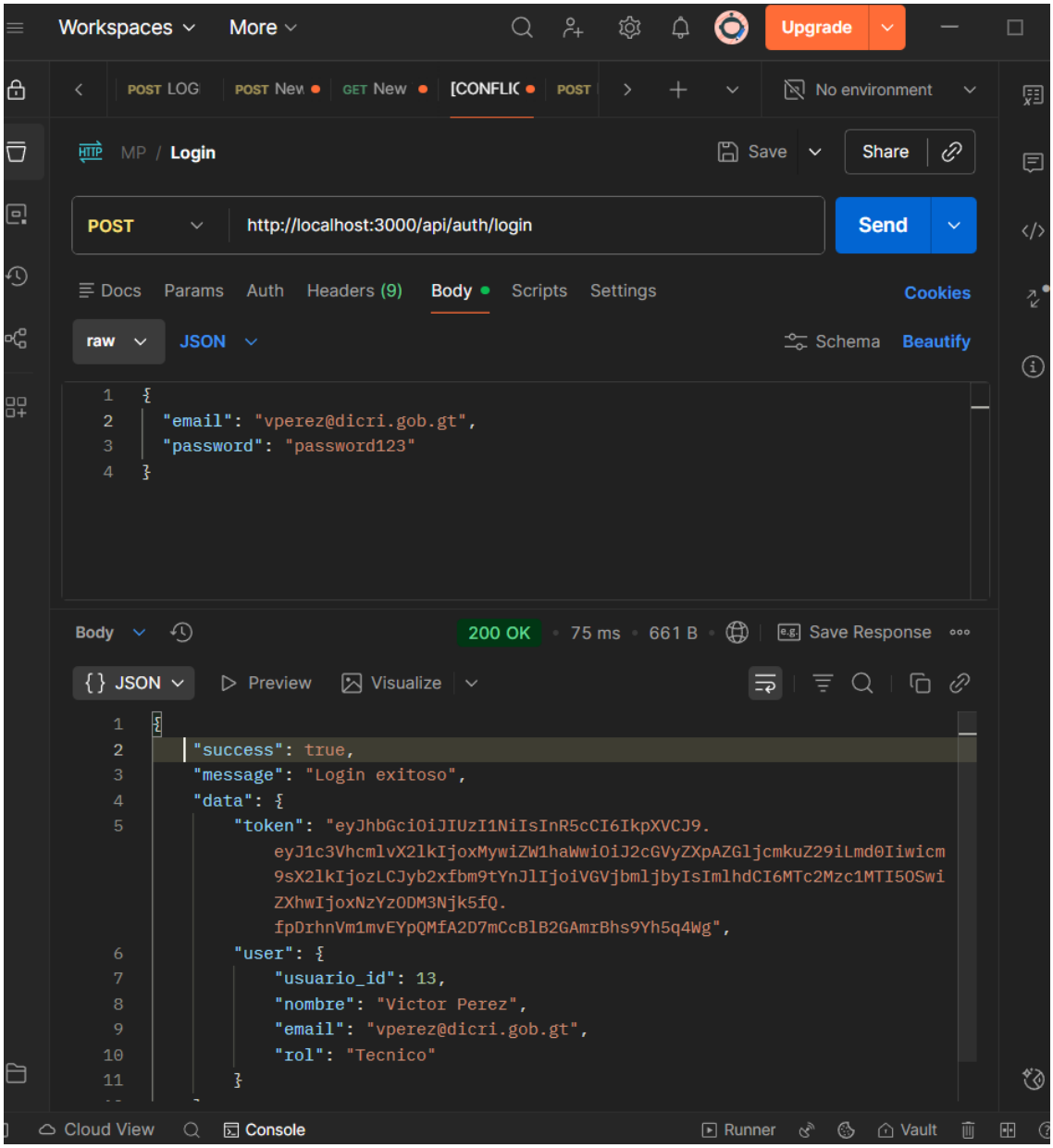
5.2.2. sp_rechazarExpediente

Aspecto	**Detalle**
Propósito	Permite a un coordinador rechazar un expediente proporcionando justificación
Parámetros	@expediente_id INT, @coordinador_id INT, @justificacion_rechazo VARCHAR(500)
Validaciones Clave	1. Solo expedientes EN_REVISION pueden ser rechazados 2. La justificación no puede estar vacía (mínimo 10 caracteres)
Actualiza	Estado → RECHAZADO, fecha_revision → GETDATE(), guarda justificación
Seguridad	Valida el estado actual antes de ejecutar la operación

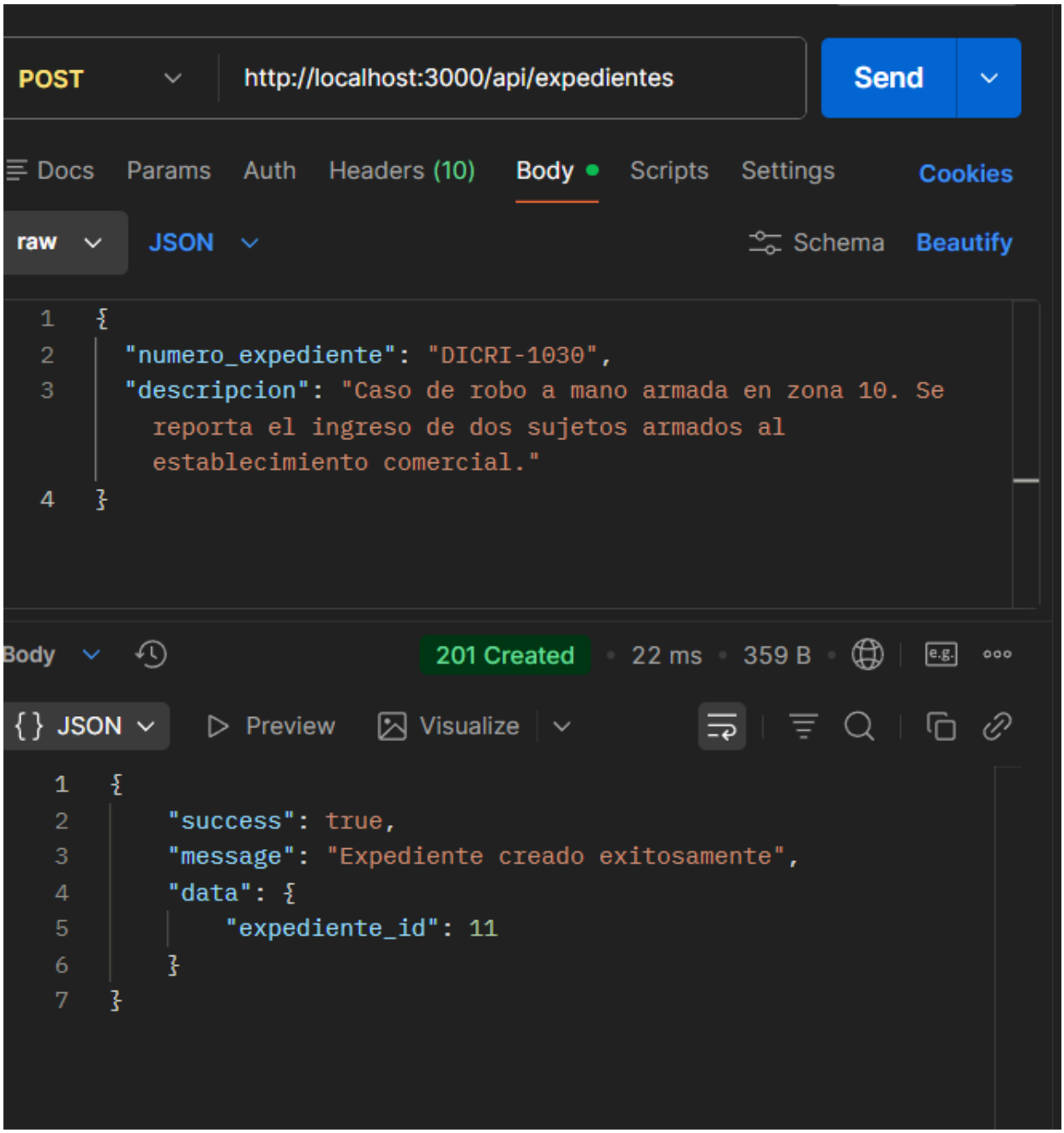
API REST – DOCUMENTACIÓN DE ENDPOINTS

6.1. Pruebas con Postman

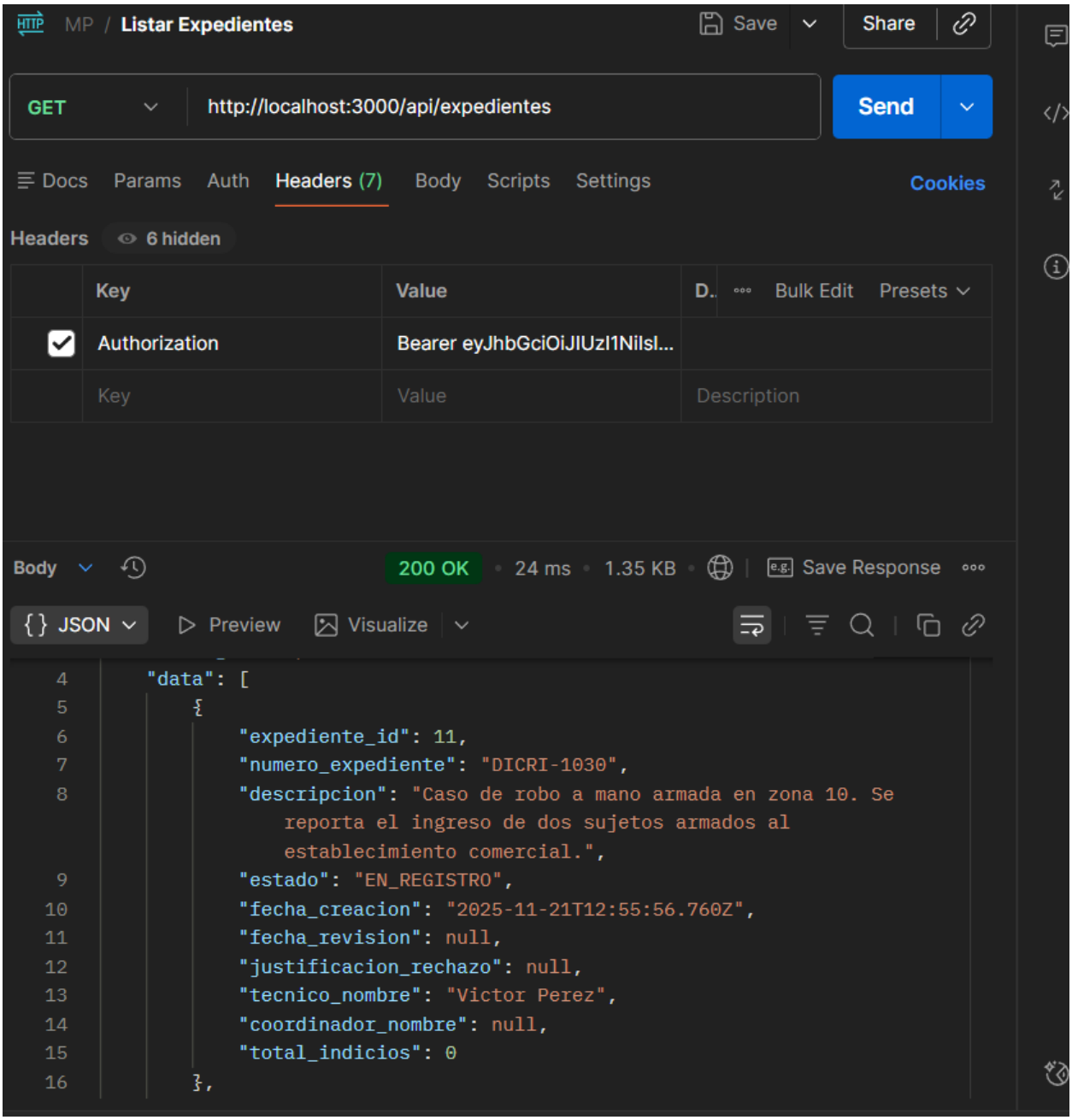
6.1.1. Login



6.1.2. Crear expediente

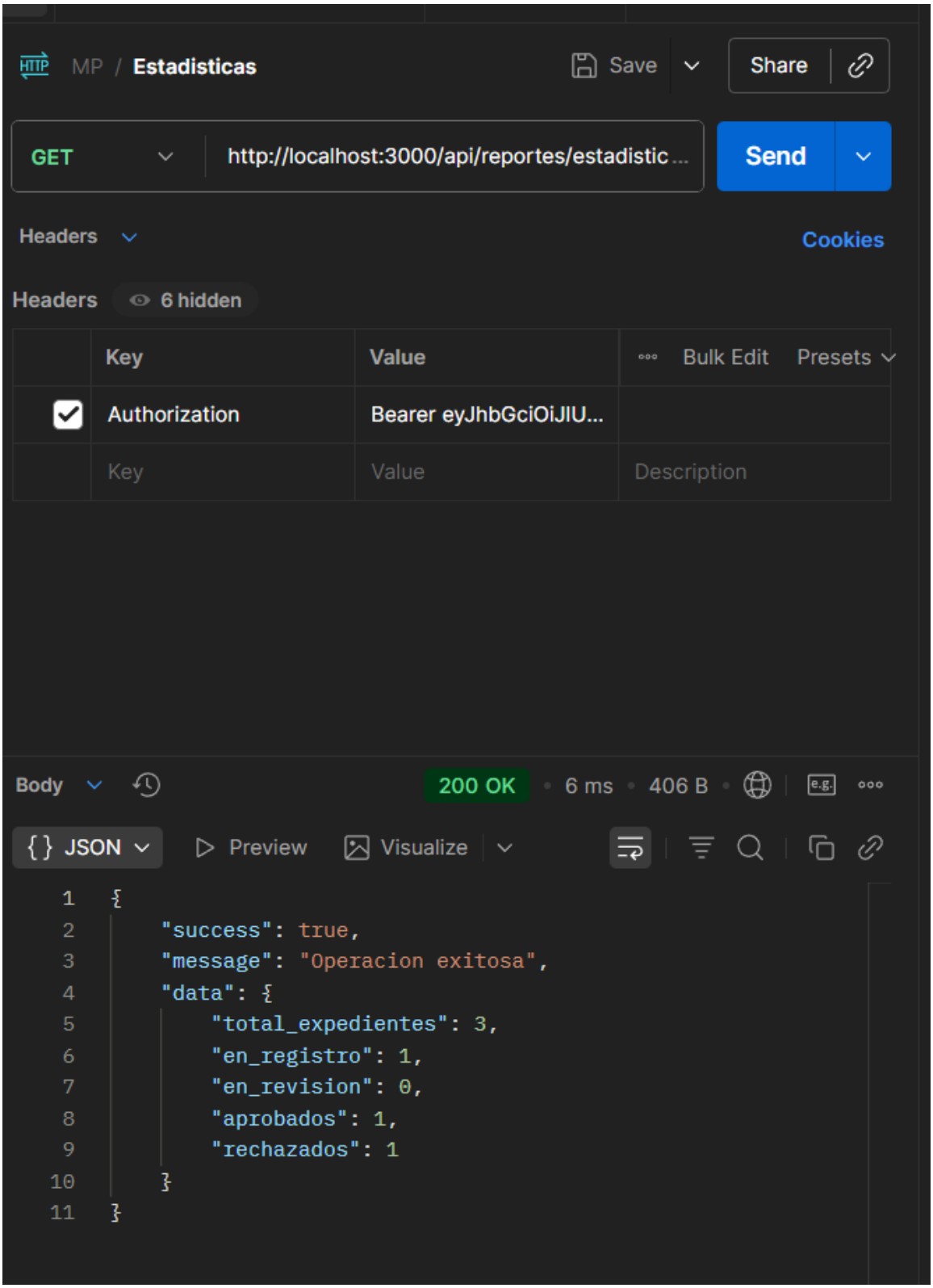


6.1.3. Listar expedientes



6.1.4. Crear indicio

6.1.5. Estadísticas



AUTENTICACIÓN Y SEGURIDAD

7.1. Sistema de autenticación JWT

El sistema utiliza JSON Web Tokens (JWT) para la autenticación. Cuando el inicio de sesión es correcto, la API genera un token con el identificador del usuario, su correo, el rol asignado y una fecha de vencimiento; ese token se devuelve al cliente y luego se envía en cada petición protegida dentro del encabezado Authorization con el formato Bearer <token>. El servidor solo verifica el token en cada solicitud, sin guardar sesión en memoria, y como el rol viaja incluido en el JWT normalmente no es necesario hacer una consulta adicional a la base de datos solo para validar permisos.

7.2. Roles y permisos

Funcionalidad	Técnico	Coordinador
Crear expedientes	✓	✗
Editar expedientes propios	✓	✗
Agregar indicios	✓	✗
Enviar a revisión	✓	✗
Aprobar expedientes	✗	✓
Rechazar expedientes	✗	✓
Ver reportes	✗	✓
Crear usuarios	✗	✗
Ver todos los expedientes	✗	Solo asignados

7.3. Hash de contraseñas

Al priorizar la seguridad del sistema, las contraseñas no pueden ir explícitamente en la base de datos como texto plano. Por lo que se utilizó bcrypt para realizar el hash y proteger la información

sensible de las cuentas. Este proceso protege al sistema de ataques de fuerza bruta, pues es casi imposible descifrar las contraseñas.

7.4. Credenciales de prueba

Rol	Email	Password
Administrador	admin@dicri.gob.gt	password123
Coordinador	jcastellanos@dicri.gob.gt	password123
Técnico	vperez@dicri.gob.gt	password123

PRUEBAS UNITARIAS

Se llevaron a cabo múltiples pruebas unitarias en el backend haciendo uso de Jest como framework de testing. Estas pruebas cubren funcionalidades críticas del sistema y asegura que los componentes principales del sistema funcionen de manera correcta al estar aislados. Se obtuvieron resultados muy satisfactorios, pues se alcanzó un 100% de éxito en todas las pruebas implementadas.

8.1. Resultados de ejecución

```
Test Suites: 4 passed, 4 total
Tests:       19 passed, 19 total
Snapshots:   0 total
Time:        1.291 s
Ran all test suites.
```

8.2. Pruebas implementadas

8.2.1. Prueba de login

```
PASS src/tests/auth.test.js
  Auth API Tests
    POST /api/auth/login
      √ debe rechazar login sin credenciales (47 ms)
      √ debe rechazar login con email invalido (8 ms)
      √ debe rechazar login con credenciales incorrectas (21 ms)
      √ debe responder a solicitudes de login (14 ms)

  console.log
    [dotenv@17.2.3] injecting env (7) from .env -- tip: ⚙ load multiple .env files with { path: ['.env.local', '.env'] }

      at _log (node_modules/dotenv/lib/main.js:142:11)
```

8.2.2. Prueba de expedientes

```
PASS src/tests/expedientes.test.js
  Expedientes API Tests
    POST /api/expedientes
      √ debe rechazar expediente sin datos (21 ms)
      √ debe rechazar sin numero_expediente (8 ms)
```

8.2.3. Prueba de utilidades

```
PASS src/tests/utils.test.js
  Response Utils Tests
    success function
      √ debe retornar respuesta exitosa con codigo 200 por defecto (2 ms)
      √ debe retornar respuesta exitosa con codigo personalizado (2 ms)
    error function
      √ debe retornar respuesta de error con codigo 500 por defecto (2 ms)
      √ debe retornar respuesta de error con codigo personalizado (1 ms)
      √ debe incluir detalles del error cuando se proporcionan (2 ms)
```

8.2.4. Prueba de validaciones

```
PASS src/tests/validation.test.js
  Validation Middleware Tests
    validateExpediente
      √ debe pasar validacion con datos correctos (4 ms)
      √ debe rechazar numero_expediente muy corto (2 ms)
      √ debe rechazar descripcion muy corta (1 ms)
    validateIndicio
      √ debe pasar validacion con datos minimos correctos (1 ms)
      √ debe rechazar sin nombre_objeto (1 ms)
    validateLogin
      √ debe pasar validacion con email y password validos (1 ms)
      √ debe rechazar email invalido
      √ debe rechazar password muy corto (1 ms)
```

DESPLIEGUE CON DOCKER

El proyecto se encuentra en contenedores de Docker, lo que permite que se pueda ejecutar y que funcione de manera consistente en cualquier ambiente. Por medio de Docker compose se levantan los tres contenedores principales, los cuales son: El frontend, el backend y el servidor de base de datos SQL Server.

9.1. Arquitectura de contenedores

Contenedor	Puerto	Función
sqlserver	1433	Motor de base de datos
backend	3000	API REST y lógica de negocio

frontend	80	Aplicación React (desarrollo)

9.2. Comandos de despliegue utilizados

9.2.1. Inicio del sistema completo

- docker-compose up -d --build

9.2.2. Detener contenedores

- docker-compose down

9.2.3. Reiniciar únicamente el backend

- docker-compose restart backend

9.2.4. Ver logs en tiempo real

- Docker-compose logs -f backend

GUÍA DE INSTALACIÓN

10.1. Requisitos

- Docker Desktop 20.10 o superior
- Git para clonar el repositorio
- 4GB de RAM disponible como mínimo
- Puerto 3000, 80 y 1433 disponibles
- 10GB de espacio de almacenamiento libre

10.2. Proceso de instalación

1. Clonar el repositorio
2. Ingresar a la carpeta del proyecto
3. Abrir la terminal dentro de la carpeta del proyecto
4. Ejecutar docker compose up
5. Acceder a la aplicación:
    a. Frontend: http://localhost:80
    b. Backend API: http://localhost:3000
    c. Health check: http://localhost:3000/health

10.3. Solución de problemas comunes

Problema	Solución
Puerto 3000 ya está en uso	Cambiar PORT en .env a otro puerto disponible (ej: 3001)
Error al conectar con SQL Server	Verificar que DB_PASSWORD coincida en .env y docker-compose.yml
Frontend no carga	Verificar que el API_URL en el frontend apunte a http://localhost:3000
Base de datos vacía	Ejecutar manualmente el script init.sql en el contenedor de SQL Server

DECISIONES TÉCNICAS Y JUSTIFICACIONES

11.1. Vite como build tool

Para el frontend se descartó Create React App y se trabajó con Vite. La razón principal fue práctica: el entorno de desarrollo arranca mucho más rápido y el recargado de módulos es casi inmediato, algo que se nota cuando se está probando varias veces el flujo de expedientes, indicios y revisión. Además, la configuración inicial es sencilla y el empaquetado final queda resuelto sin necesidad de ajustar demasiados parámetros.

11.2. TailwindCSS para estilos

Se decidió que la implementación de TailwindCSS era lo indicado en lugar de usar hojas de estilo tradicionales, esto debido a que permite trabajar con clases utilitarias directamente en los componentes, lo que facilita ajustar márgenes, tamaños y colores sin necesidad de armar un framework de estilos propio.

11.3. Uso de Docker con volúmenes persistentes

En la configuración con Docker se decidió guardar la información de SQL Server en un volumen. Esto con el objetivo de que los expedientes y los indicios no se borren cada vez que se apaga o se vuelve a levantar el contenedor, sino que queden guardados, aunque el servicio se reinicie varias veces durante las pruebas.

En el archivo docker-compose.yml se declaró un volumen y se vinculó con la carpeta donde SQL Server deja sus archivos de datos. Gracias a eso, el contenedor se puede recrear o actualizar la imagen de la aplicación sin tener que cargar de nuevo toda la base desde cero.

Con este esquema, el entorno de pruebas se comporta de forma más parecida a un ambiente real, en el que la información del sistema de evidencias debe mantenerse, aunque haya reinicios o cambios en los servicios.

11.4. Ausencia de módulo gráfico para registro de usuarios

No se incluyó un apartado específico para registrar usuarios nuevos. Esto se debe a dos causas principales:

- Priorización de funcionalidad: con el plazo de la prueba se dio prioridad al flujo principal (expedientes, indicios, revisión y reportes) por encima de tareas administrativas
- Soporte desde backend: el sistema sí cuenta con un endpoint protegido que permite crear usuarios desde el lado del servidor (POST /api/auth/register), pensado para uso de un administrador. Para la demostración se cargan usuarios de ejemplo con los distintos roles.

En un despliegue productivo, lo natural sería añadir un módulo de administración donde el rol de Administrador pueda gestionar cuentas y permisos desde la interfaz.

11.5. Credenciales visibles en el formulario de inicio de sesión

Durante las pruebas internas se vio necesario que el evaluador pudiera cambiar rápido entre un usuario técnico, un coordinador y un administrador. Por ese motivo, en el contexto de esta prueba se optó por dejar las credenciales de ejemplo claramente disponibles (en el manual y en el formulario de login).

Se asume explícitamente que esta práctica solo es válida en un ambiente de demostración. En producción, los usuarios deben contar con su propia contraseña y obviamente las credenciales no deben mostrarse en pantalla. Para perfiles sensibles (por ejemplo, coordinadores o administradores) sería recomendable complementar con mecanismos adicionales, como doble factor de autenticación.

Con estas aclaraciones se busca dejar separado lo que se hizo para facilitar la evaluación técnica y lo que se consideraría apropiado en un entorno real.

CONCLUSIONES

El sistema desarrollado cubre lo que se pidió en la prueba técnica y permite trabajar correctamente con las evidencias. Se cuenta con autenticación basada en tres perfiles (Administrador, Coordinador y Técnico), registro y manejo de expedientes e indicios con cambio de estado controlado, un proceso de revisión donde el coordinador puede aprobar o rechazar y, en caso de rechazo, debe dejar una justificación, además de un apartado de reportes que admite filtrar por distintos criterios de consulta.

Internamente se organizó la solución en tres capas (frontend, API y base de datos), apoyándose en diecisiete procedimientos almacenados para todo el acceso a datos y en un conjunto de diecinueve pruebas unitarias que se ejecutan correctamente sobre la lógica principal.