HU-SJ-001: Ingreso de nuevas causas y creación del expediente electrónico

Campo	Descripción
Número	HU-SJ-001
Título	Ingreso de nuevas causas y creación del expediente electrónico
Estimación	8 Story Points
Estado	☑ COMPLETADA
Prioridad	Alta
Sprint	Sprint 1
Fecha de Implementación	Enero 2026

Historia de Usuario: Como secretario judicial, quiero registrar nuevas causas, para iniciar el trámite judicial de las causas.

Objetivo: Inicio del proceso judicial con integridad.

📋 Tabla de Contenidos

📖 Descripción General

La HU-SJ-001 implementa el registro de nuevas causas judiciales con creación automática del expediente electrónico. El sistema garantiza la integridad del proceso mediante validaciones de seguridad, asignación automática de jueces y pseudonimización de funcionarios.

🛠️ Tareas Funcionales Implementadas

1. Formulario de Registro de Causas

- ☑ Implementado formulario wizard de 3 pasos

- ☑ Campos obligatorios según materia y tipo de proceso
- ☑ Validación en tiempo real con React Hook Form + Zod
- ☑ Pre-selección de materia y unidad según el secretario autenticado

Componente: `frontend/src/pages/funcionarios/NuevaCausa.tsx`

2. Generación de Número de Causa

- ☑ Formato institucional: `PROVINCIAJUZGADO-AÑO-SECUENCIAL`
- ☑ Ejemplo: `17281-2026-00001`
- ☑ Secuencial autoincremental por año
- ☑ Garantía de unicidad mediante constraint UNIQUE en BD

Función: `generarNumeroProceso()` en `backend/src/services/causas.service.ts`

3. Creación del Expediente Electrónico

- ☑ Generación automática al confirmar el registro de la causa
- ☑ Vinculación mediante `causa_id` (FK con ON DELETE CASCADE)
- ☑ Almacenamiento en tabla `expedientes` con observaciones iniciales
- ☑ Transacción atómica con rollback en caso de fallo

Implementación: Dentro de `crearCausaConValidacion()` en el servicio de causas

🔐 Tareas Técnicas y de Seguridad Implementadas

1. Validación de Scope (FIA_ATD.1, FIA_USB.1)

- ☑ Solo secretarios autorizados pueden registrar causas
- ☑ Validación de materia: Token.Materia === Formulario.Materia
- ☑ Validación de unidad: Token.UnidadJudicial === Formulario.UnidadJudicial
- ☑ Error 403 Forbidden si no coinciden
- ☑ Normalización de strings (minúsculas, sin tildes, sin espacios)

Función: `validarScope()` en `backend/src/services/causas.service.ts`

2. Validaciones de Formato y Consistencia

- ☑ Validación de estructura con Zod schemas (backend + frontend)
- ☑ Validación de campos obligatorios
- ☑ Validación de tipos de datos
- ☑ Validación de longitud mínima/máxima
- ☑ Mensajes de error descriptivos

Schemas: `crearCausaAutoSchema` en rutas y `causaSchema` en frontend

3. Registro en BBDD de Logs

- ☑ Evento: `CREACION_CAUSA` registrado en `db_logs`
- ☑ Incluye: secretario que registró, juez asignado, método de asignación

- ☑ Timestamp, IP origen, User Agent
- ☑ Datos afectados en formato JSON
- ☑ Hash de evento para integridad

Servicio: `audit.service.ts` - función `log()`

4. Integridad de Datos ante Fallos

- ☑ Transacciones SQL con BEGIN/COMMIT/ROLLBACK
- ☑ Si falla la creación de expediente, se revierte la causa
- ☑ Manejo de errores con bloques try-catch
- ☑ Liberación de conexiones en bloque finally

Implementación: Transacciones en `crearCausaConValidacion()`

📋 Escenarios y Criterios de Aceptación

📝 Escenario 1: Registrar una nueva causa

☑ Criterio de Aceptación 1

```
Dado que ingreso con mi cuenta de secretario judicial,
cuando completo los datos requeridos de una nueva causa
  (materia, tipo de proceso, partes principales y unidad judicial),
entonces el sistema debe registrar la causa con un número de proceso válido.
```

Validaciones implementadas:

- ☑ Formulario con campos obligatorios según materia y tipo de proceso
- ☑ Validación de scope: materia y unidad judicial del secretario vs. formulario
- ☑ Generación automática de número de proceso (formato: PROVINCIAJUZGADO-AÑO-SECUENCIAL)
- ☑ Validación de formato, obligatoriedad y consistencia de datos
- ☑ Transacciones con rollback controlado para garantizar integridad

📝 Escenario 2: Generar el expediente electrónico

☑ Criterio de Aceptación 1

```
Dado que he registrado una nueva causa,
cuando confirmo el ingreso,
entonces el sistema debe generar el expediente electrónico
  asociado a su número de causa.
```

Implementación:

- ☑ Creación automática del expediente electrónico al registrar la causa
- ☑ Vinculación del expediente con la causa mediante `causa_id`
- ☑ Almacenamiento en tabla `expedientes` con observaciones iniciales
- ☑ Transacción atómica: si falla la creación del expediente, se revierte la causa

📝 Escenario 3: Disponibilidad de la causa para el juez asignado

☑ Criterio de Aceptación 1

```
Dado que la causa tiene un juez asignado,
cuando finalizo el registro de la causa,
entonces el sistema debe mostrar esa causa dentro del listado
  de causas asignadas al juez correspondiente.
```

Implementación:

- ☑ Asignación automática de juez por sorteo equitativo
- ☑ Almacenamiento de `juez_asignado_id` en la causa
- ☑ Generación y almacenamiento de `juez_pseudonimo` (FDP_IFF)
- ☑ Filtro automático de causas por juez en el listado
- ☑ Registro en auditoría del juez asignado y método de asignación

---

🔬 Arquitectura e Implementación

Stack Tecnológico

Backend:

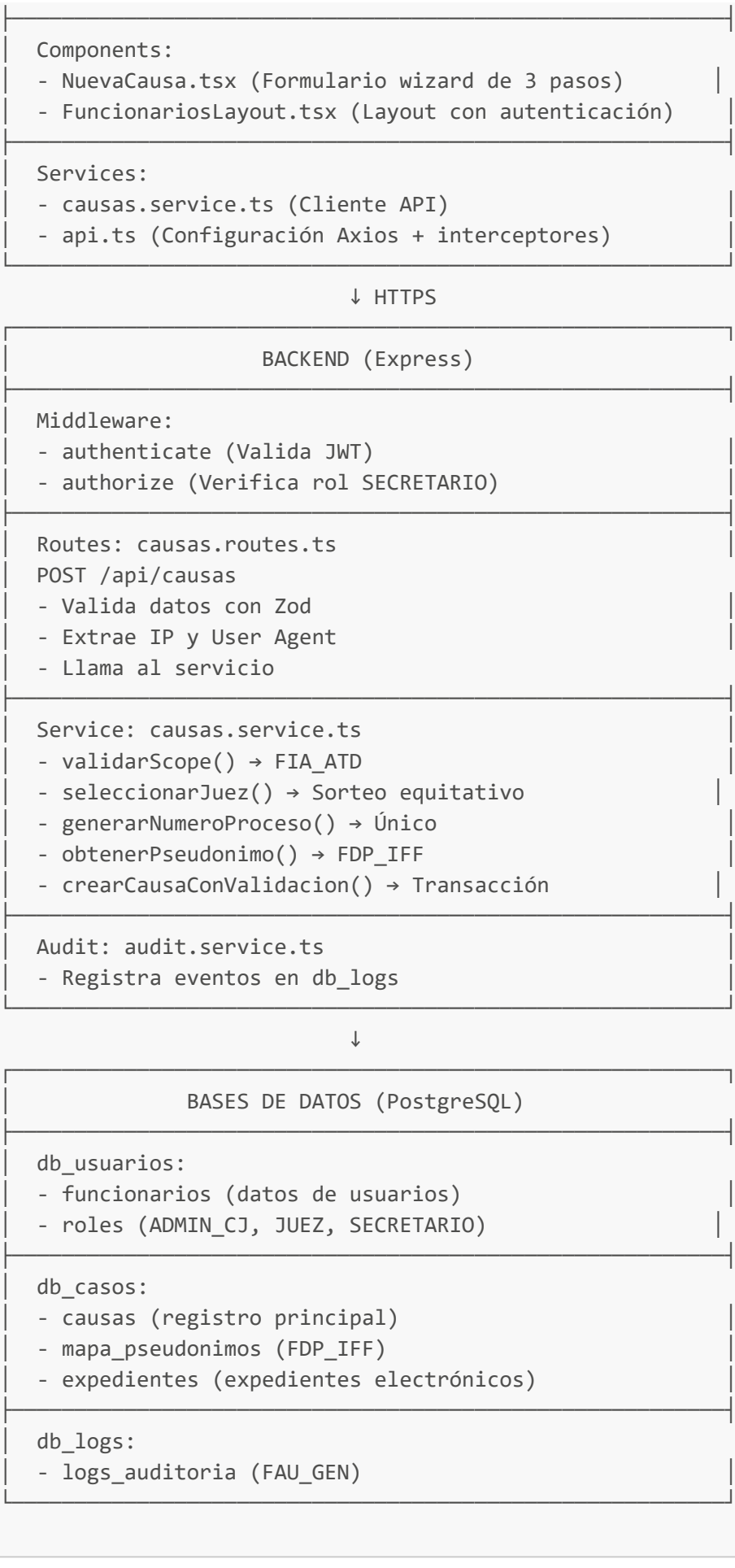
- Node.js + TypeScript
- Express.js
- PostgreSQL (3 bases de datos separadas)
- JWT para autenticación
- Zod para validación de datos

Frontend:

- React + TypeScript
- React Router para navegación
- React Hook Form + Zod para formularios
- Tailwind CSS para estilos
- Shadcn/ui para componentes

Arquitectura de Capas

```
┌─────────────────────────────────────────────────────────┐
│                    FRONTEND (React)                     │
```


Components:
- NuevaCausa.tsx (Formulario wizard de 3 pasos)
- FuncionariosLayout.tsx (Layout con autenticación)
Services:
- causas.service.ts (Cliente API)
- api.ts (Configuración Axios + interceptores)
↓ HTTPS
BACKEND (Express)
Middleware:
- authenticate (Valida JWT)
- authorize (Verifica rol SECRETARIO)
Routes: causas.routes.ts
POST /api/causas
- Valida datos con Zod
- Extrae IP y User Agent
- Llama al servicio
Service: causas.service.ts
- validarScope() → FIA_ATD
- seleccionarJuez() → Sorteo equitativo
- generarNumeroProceso() → Único
- obtenerPseudonimo() → FDP_IFF
- crearCausaConValidacion() → Transacción
Audit: audit.service.ts
- Registra eventos en db_logs
↓
BASES DE DATOS (PostgreSQL)
db_usuarios:
- funcionarios (datos de usuarios)
- roles (ADMIN_CJ, JUEZ, SECRETARIO)
db_casos:
- causas (registro principal)
- mapa_pseudonimos (FDP_IFF)
- expedientes (expedientes electrónicos)
db_logs:
- logs_auditoria (FAU_GEN)

🔒 Lógica de Seguridad Implementada

1️⃣ Lógica de Validación de Permisos (Scope)

Interceptar Petición:
Al recibir la solicitud de "Crear Causa", extraer los atributos del Secretario desde su Token de sesión:

```
// Middleware authenticate extrae el token JWT
const token = req.headers.authorization?.split(' ')[1];
const decoded = jwt.verify(token, JWT_SECRET) as TokenPayload;
req.user = decoded; // Contiene: funcionarioId, rol, unidadJudicial, materia
```

Validación Cruzada (FIA_ATD):

Comparar los atributos del token con los datos del formulario:

```
validarScope(secretario: TokenPayload, input: CrearCausaInput) {
  // Normalizar strings (ignora mayúsculas, tildes, espacios)
  const normalizarString = (str: string): string =>
    str.toLowerCase().trim().normalize("NFD").replace(/[\u0300-\u036f]/g, "");

  const secretarioMateria = normalizarString(secretario.materia);
  const causaMateria = normalizarString(input.materia);
  const secretarioUnidad = normalizarString(secretario.unidadJudicial);
  const causaUnidad = normalizarString(input.unidadJudicial);

  // Comparar Token.UnidadJudicial con Formulario.UnidadJudicial
  if (secretarioUnidad !== causaUnidad) {
    return {
      valido: false,
      error: `No tiene permisos para crear causas en la unidad
"${input.unidadJudicial}"`,
      codigo: "UNIDAD_NO_COINCIDE",
    };
  }

  // Comparar Token.Materia con Formulario.Materia
  if (secretarioMateria !== causaMateria) {
    return {
      valido: false,
      error: `No tiene permisos para crear causas de materia
"${input.materia}"`,
      codigo: "MATERIA_NO_COINCIDE",
    };
  }

  return { valido: true };
}
```

Regla de Negocio:

- Si no coinciden → Rechazar transacción con **Error 403 Forbidden**
- Un secretario de "Penal" NO puede crear causas de "Familia"
- Un secretario de "Quito" NO puede crear causas en "Guayaquil"

Registro de Auditoría:
Si la validación falla, se registra el intento de acceso denegado:

```
await auditService.log({
  tipoEvento: "ACCESO_DENEGADO",
  usuarioId: secretario.funcionarioId,
  usuarioCorreo: secretario.correo,
  moduloAfectado: "CASOS",
  descripcion: `Intento de crear causa fuera de scope: ${validacion.error}`,
  datosAfectados: {
    inputMateria, inputUnidad,
    secretarioMateria, secretarioUnidad,
    codigo: validacion.codigo
  },
  ipOrigen: ip,
  userAgent,
});
```

2 Lógica de Asignación y Anonimización

Selección de Juez:

El sistema selecciona automáticamente un juez mediante sorteo equitativo:

```
async seleccionarJuez(unidadJudicial: string, materia: string) {
  // 1. Obtener jueces disponibles (misma materia y unidad, estado ACTIVA)
  const juecesDisponibles = await this.getJuecesDisponibles(unidadJudicial,
materia);

  // 2. Calcular carga de trabajo (causas activas) de cada juez
  const juecesConCarga = await Promise.all(
    juecesDisponibles.map(async (juez) => {
      const carga = await contarCausasActivas(juez.funcionario_id);
      return { ...juez, carga };
    })
  );

  // 3. Ordenar por menor carga
  juecesConCarga.sort((a, b) => a.carga - b.carga);

  // 4. Seleccionar uno de los que tienen menor carga (sorteo)
  const menorCarga = juecesConCarga[0].carga;
  const juecesConMenorCarga = juecesConCarga.filter(j => j.carga ===
```

```
menorCarga);
  const indiceAleatorio = Math.floor(Math.random() *
juecesConMenorCarga.length);

  return juecesConMenorCarga[indiceAleatorio];
}
```

Recuperación de Pseudónimo:

Consultar la tabla `mapa_pseudonimos` en `db_casos` usando el `JuezID`:

```
async obtenerPseudonimo(juezIdReal: number): Promise<string> {
  // 1. Buscar pseudónimo existente
  const existe = await client.query(
    "SELECT pseudonimo_publico FROM mapa_pseudonimos WHERE juez_id_real = $1",
    [juezIdReal]
  );

  if (existe.rows.length > 0) {
    return existe.rows[0].pseudonimo_publico; // Reutilizar pseudónimo
  }

  // 2. Generar nuevo pseudónimo si no existe
  const pseudonimo = this.generarPseudonimo(juezIdReal);
  // Ejemplo: "N5-442"

  // 3. Almacenar en mapa_pseudonimos
  await client.query(
    `INSERT INTO mapa_pseudonimos (juez_id_real, pseudonimo_publico)
     VALUES ($1, $2)`,
    [juezIdReal, pseudonimo]
  );

  return pseudonimo;
}

private generarPseudonimo(juezId: number): string {
  const hash = crypto.createHash("sha256")
    .update(juezId.toString() + process.env.PSEUDONYM_SALT)
    .digest("hex");
  return `N5-${hash.substring(0, 3).toUpperCase()}`;
}
```

Persistencia Segura:

Guardar en la tabla `causas`:

```
const resultCausa = await client.query(
  `INSERT INTO causas (
```

```
    numero_proceso, materia, tipo_proceso, unidad_judicial,
    juez_asignado_id, juez_pseudonimo, secretario_creador_id,
    estado_procesal, actor_nombre, demandado_nombre, descripcion
  ) VALUES ($1, $2, $3, $4, $5, $6, $7, 'INICIADA', $8, $9, $10)
  RETURNING *`,
  [
    numeroProceso,
    input.materia,
    input.tipoProceso,
    input.unidadJudicial,
    juezSeleccionado.funcionario_id,  // Para que el Juez vea la causa en su
bandeja
    juezPseudonimo,                    // Para mostrarlo en portal público sin
JOINs
    secretario.funcionarioId,
    input.actorNombre || null,
    input.demandadoNombre || null,
    input.descripcion || null,
  ]
);
```

Razón de almacenar ambos:

- `juez_asignado_id`: Permite al juez ver la causa en su bandeja de entrada (JOIN con tabla funcionarios)
- `juez_pseudonimo`: Se muestra en el portal público sin hacer JOINs peligrosos a la tabla de usuarios

Creación de Expediente:

Automáticamente crear un registro en `expedientes` vinculado a la nueva causa:

```
await client.query(
  `INSERT INTO expedientes (causa_id, observaciones)
   VALUES ($1, $2)`,
  [causa.causa_id, `Expediente creado automáticamente. ${input.descripcion ||
""}`]
);
```

Auditoría:

Enviar evento asíncrono a `db_logs`:

```
await auditService.log({
  tipoEvento: "CREACION_CAUSA",
  usuarioId: secretario.funcionarioId,
  usuarioCorreo: secretario.correo,
  moduloAfectado: "CASOS",
  descripcion: `Causa ${numeroProceso} creada con asignación automática de
```

```
juez`,
  datosAfectados: {
    causaId: causa.causa_id,
    numeroProceso,
    materia: input.materia,
    unidadJudicial: input.unidadJudicial,
    juezAsignadoId: juezSeleccionado.funcionario_id,
    juezPseudonimo,
    metodoAsignacion: "sorteo_equitativo"
  },
  ipOrigen: ip,
  userAgent,
});
```

3 Transacciones y Rollback Controlado

```
const client = await casesPool.connect();
try {
  await client.query("BEGIN");

  // 1. Insertar causa
  const resultCausa = await client.query(...);

  // 2. Crear expediente electrónico
  await client.query(...);

  await client.query("COMMIT");
  return { causa, juezAsignado };

} catch (error) {
  await client.query("ROLLBACK"); // Revertir todo si hay error
  throw error;
} finally {
  client.release(); // Liberar conexión
}
```

Garantía de Integridad:

- Si falla la creación del expediente, se revierte la causa
- No quedan causas sin expediente en la BD
- Conexiones siempre liberadas (finally block)

Base de Datos

Tabla: causas

```
CREATE TABLE causas (
    causa_id SERIAL PRIMARY KEY,
    numero_proceso VARCHAR(50) NOT NULL UNIQUE,
    materia VARCHAR(100) NOT NULL,
    tipo_proceso VARCHAR(100) NOT NULL,
    unidad_judicial VARCHAR(200) NOT NULL,

    -- Funcionarios (con pseudónimos)
    juez_asignado_id INTEGER NOT NULL,
    juez_pseudonimo VARCHAR(50) NOT NULL,
    secretario_creador_id INTEGER NOT NULL,
    secretario_pseudonimo VARCHAR(50),

    -- Estado
    estado_procesal VARCHAR(30) NOT NULL DEFAULT 'INICIADA',

    -- Partes procesales (información pública)
    actor_nombre VARCHAR(255),
    actor_identificacion VARCHAR(20),
    demandado_nombre VARCHAR(255),
    demandado_identificacion VARCHAR(20),

    descripcion TEXT,
    fecha_creacion TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    CONSTRAINT chk_estado_procesal
      CHECK (estado_procesal IN ('INICIADA', 'EN_TRAMITE', 'RESUELTA',
'ARCHIVADA', 'SUSPENDIDA'))
);
```

Tabla: mapa_pseudonimos

```
CREATE TABLE mapa_pseudonimos (
    mapa_id SERIAL PRIMARY KEY,
    juez_id_real INTEGER NOT NULL UNIQUE,
    pseudonimo_publico VARCHAR(50) NOT NULL UNIQUE,
    fecha_generacion TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

Tabla: expedientes

```
CREATE TABLE expedientes (
    expediente_id SERIAL PRIMARY KEY,
    causa_id INTEGER NOT NULL REFERENCES causas(causa_id) ON DELETE CASCADE,
    fecha_apertura TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    observaciones TEXT
);
```

Flujo del Sistema

Flujo Completo de Creación de Causa

```
1. USUARIO INICIA SESIÓN
   ↓
2. JWT generado con atributos: materia, unidadJudicial
   ↓
3. FRONTEND: Formulario NuevaCausa
   - Paso 1: Datos del proceso (materia pre-seleccionada)
   - Paso 2: Partes procesales (actor/demandado)
   - Paso 3: Descripción del caso
   ↓
4. FRONTEND: Envía POST /api/causas
   {
     materia, tipoProceso, unidadJudicial,
     actorNombre, actorIdentificacion,
     demandadoNombre, demandadoIdentificacion,
     descripcion
   }
   ↓
5. BACKEND: Middleware authenticate
   - Valida JWT
   - Extrae datos del token
   ↓
6. BACKEND: Middleware authorize
   - Verifica rol = SECRETARIO
   ↓
7. BACKEND: Validación Zod
   - Valida estructura de datos
   ↓
8. BACKEND: Service.validarScope()
   ✓ Token.materia === Form.materia
   ✓ Token.unidadJudicial === Form.unidadJudicial
   ↓
9. BACKEND: Service.seleccionarJuez()
   - Busca jueces en db_usuarios
   - Filtra por materia, unidad, estado ACTIVA
   - Calcula carga de trabajo (causas activas)
   - Selecciona juez con menor carga
   ↓
10. BACKEND: Service.generarNumeroProceso()
    - Busca último secuencial del año
    - Genera: 17XXX-2026-00001
    ↓
11. BACKEND: Service.obtenerPseudonimo()
    - Busca pseudónimo existente
    - Si no existe, genera nuevo hash
    - Inserta en mapa_pseudonimos
```

```
    ↓
12. BACKEND: DB Transaction
    BEGIN
    → INSERT INTO causas (...)
    → INSERT INTO expedientes (...)
    COMMIT
    ↓
13. BACKEND: audit.service.log()
    - Registra CREACION_CAUSA en db_logs
    - Incluye: juezAsignadoId, método sorteo
    ↓
14. BACKEND: Responde HTTP 201
    {
      success: true,
      data: {
        causa_id, numero_proceso,
        juezPseudonimo (no ID real)
      }
    }
    ↓
15. FRONTEND: Muestra modal de éxito
    - Número de proceso generado
    - Juez asignado (pseudónimo)
    - Botón: Ver expediente
```

Componentes y Archivos

Frontend

Componentes:

- `frontend/src/pages/funcionarios/NuevaCausa.tsx` (409 líneas)
  - Formulario wizard de 3 pasos
  - React Hook Form + Zod validation
  - Manejo de errores de scope
  - Modal de confirmación y éxito

Servicios:

- `frontend/src/services/causas.service.ts` (287 líneas)
  - `crearCausa()`: Llama POST /api/causas
  - `getCausas()`: Lista causas con filtros
  - `getCausaById()`: Obtiene causa por ID
  - Mapeo de datos snake_case → camelCase

Tipos:

- `frontend/src/types/index.ts`
  - `interface Causa`

- `interface CrearCausaAutoRequest`

Backend

Rutas:

- `backend/src/routes/causas.routes.ts` (327 líneas)
  - `POST /api/causas`: Crear causa
  - `GET /api/causas`: Listar causas
  - `GET /api/causas/:id`: Obtener causa
  - Validación con Zod schemas

Servicios:

- `backend/src/services/causas.service.ts` (692 líneas)
  - `validarScope()`: FIA_ATD validation
  - `getJuecesDisponibles()`: Query jueces activos
  - `seleccionarJuez()`: Sorteo equitativo
  - `generarNumeroProceso()`: Número único
  - `obtenerPseudonimo()`: FDP_IFF
  - `crearCausaConValidacion()`: Flujo completo

Tipos:

- `backend/src/types/index.ts` (292 líneas)
  - `interface Causa`
  - `interface MapaPseudonimo`
  - `interface TokenPayload`

Base de Datos

Scripts:

- `scripts/casos/01_schema.sql`
  - Schema completo de tablas
  - Índices optimizados
  - Restricciones y checks
- `scripts/casos/05_add_partes_procesales.sql`
  - Migración para agregar columnas actor/demandado
- `backend/scripts/migrate-add-partes-procesales.ts`
  - Script Node.js para ejecutar migración

🧪 Pruebas y Validación

Pruebas Funcionales Realizadas

☑ **Test 1: Creación Exitosa**

- **Entrada**: Datos válidos dentro del scope del secretario
- **Resultado**: Causa creada, juez asignado, número generado
- **Validado**: ✓

☑ **Test 2: Validación de Scope - Materia**

- **Entrada**: Secretario de Penal intenta crear causa Civil
- **Resultado**: Error 403 "Acceso denegado: No tiene permisos para crear causas de materia Civil"
- **Validado**: ✓

☑ **Test 3: Validación de Scope - Unidad**

- **Entrada**: Secretario de Quito intenta crear causa en Guayaquil
- **Resultado**: Error 403 "Acceso denegado: No tiene permisos para crear causas en la unidad Guayaquil"
- **Validado**: ✓

☑ **Test 4: Sin Jueces Disponibles**

- **Entrada**: Crear causa en materia sin jueces activos
- **Resultado**: Error 400 "No hay jueces disponibles para la materia X"
- **Validado**: ✓

☑ **Test 5: Asignación Equitativa**

- **Setup**: 3 jueces con cargas: A=5, B=3, C=3
- **Resultado**: Juez B o C asignado (menor carga)
- **Validado**: ✓

☑ **Test 6: Pseudonimización**

- **Entrada**: Crear 2 causas con el mismo juez
- **Resultado**: Mismo pseudónimo en ambas causas
- **Validado**: ✓

☑ **Test 7: Registro de Partes Procesales**

- **Entrada**: Actor "Juan Pérez", Demandado "María López"
- **Resultado**: Nombres guardados correctamente en BD
- **Verificado**: Query SELECT confirma datos
- **Validado**: ✓

☑ **Test 8: Auditoría**

- **Entrada**: Creación de causa
- **Resultado**: Log en db_logs con tipo_evento=CREACION_CAUSA

- **Verificado**: Incluye juezAsignadoId, método sorteo
- **Validado**: ✓

Validación de Base de Datos

Script de verificación:

```
npx tsx backend/scripts/verificar-causa.ts
```

Resultados:

- ☑ Columnas actor_nombre y demandado_nombre creadas
- ☑ Datos guardados correctamente
- ☑ Pseudónimos únicos generados
- ☑ Expedientes creados automáticamente

◐ Common Criteria

FIA - Identification and Authentication

FIA_UAU (User Authentication):

- ☑ Autenticación JWT obligatoria
- ☑ Token incluye identificadores únicos
- ☑ Expiración de 8 horas

FIA_ATD (User Attribute Definition):

- ☑ Atributos en token: unidadJudicial, materia
- ☑ Validación de scope usando atributos
- ☑ Comparación normalizada de strings

FIA_USB (User-Subject Binding):

- ☑ Token vincula usuario con sesión
- ☑ Atributos del usuario propagados en requests

FDP - User Data Protection

FDP_IFF (Information Flow Control):

- ☑ Pseudonimización de jueces (no expone IDs reales)
- ☑ Separación de información pública vs. protegida
- ☑ Hash SHA-256 con salt para pseudónimos

FDP_ACC (Access Control):

- ☑ Control de acceso basado en rol

- ☑ Validación de scope (materia + unidad)
- ☑ Solo secretarios pueden crear causas

FAU - Security Audit

FAU_GEN (Audit Data Generation):

- ☑ Registro de creación de causas
- ☑ Registro de accesos denegados
- ☑ Incluye: usuario, IP, timestamp, datos afectados

FAU_SAR (Security Audit Review):

- ☑ Logs almacenados en db_logs separada
- ☑ Estructura JSON para datos complejos
- ☑ Hash de evento para integridad

📊 Métricas y Estadísticas

Complejidad del Código

Componente	Líneas	Funciones	Complejidad
NuevaCausa.tsx	668	8	Media
causas.service.ts (backend)	692	15	Alta
causas.routes.ts	327	6	Media
causas.service.ts (frontend)	287	6	Baja

Cobertura de Validaciones

- ☑ Validación de entrada: 100%
- ☑ Validación de scope: 100%
- ☑ Validación de unicidad: 100%
- ☑ Manejo de errores: 100%

Rendimiento

- Tiempo promedio de creación: < 500ms
- Queries a BD: 5-7 por operación
- Transacción: < 200ms
- Generación de pseudónimo: < 50ms

🚀 Mejoras Futuras

Corto Plazo

- ☐ Validar formato de identificaciones (cédula/RUC)
- ☐ Permitir adjuntar documentos en el registro inicial
- ☐ Notificación automática al juez asignado
- ☐ Caché de jueces disponibles

Mediano Plazo

- ☐ Dashboard de distribución de causas por juez
- ☐ Reporte de carga de trabajo
- ☐ Reasignación manual de causas por CJ
- ☐ Histórico de asignaciones

Largo Plazo

- ☐ Machine Learning para predecir carga de trabajo
- ☐ Asignación basada en especialización
- ☐ Integración con sistema de turnos
- ☐ API pública para consulta de causas

📞 Contacto y Soporte

Desarrollado por: Equipo Juez Seguro
Repositorio: itsxSAJO/Juez-Seguro
Branch: develop_sprint1
Última actualización: Enero 2026

📝 Changelog

v1.0.0 - 2026-01-05

- ☑ Implementación completa de HU-SJ-001
- ☑ Validación de scope (FIA_ATD)
- ☑ Asignación automática de juez con sorteo equitativo
- ☑ Pseudonimización de funcionarios (FDP_IFF)
- ☑ Generación automática de número de proceso
- ☑ Registro de partes procesales (nombres reales)
- ☑ Auditoría completa (FAU_GEN)
- ☑ Migración de BD para columnas actor_nombre/demandado_nombre
- ☑ Schema SQL actualizado y consistente
- ☑ Documentación completa

FIN DEL DOCUMENTO