HU-CJ-001: Registro y Administración de Cuentas

📋 Información General

Campo	Valor
ID	HU-CJ-001
Título	Registro y Administración de Cuentas
Sprint	Sprint 1
Objetivo	Gestión segura de identidades según Common Criteria (FIA)
Actor Principal	Administrador del Consejo de la Judicatura (ADMIN_CJ)
Última Actualización	5 de enero de 2026

🎯 Descripción

Esta historia de usuario implementa el sistema de gestión de identidades para el sistema judicial "Juez Seguro", cumpliendo con los requisitos de seguridad de Common Criteria:

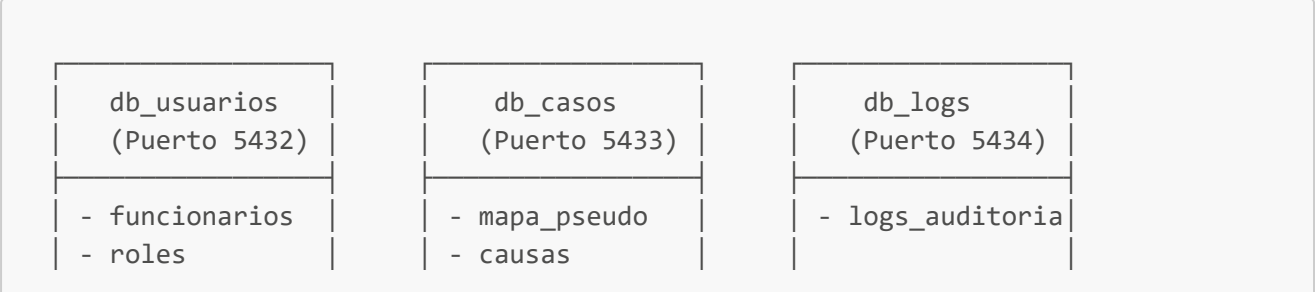
- **FIA_UID** - Identificación de Usuario
- **FIA_ATD** - Definición de Atributos de Usuario
- **FIA_AFL** - Manejo de Fallos de Autenticación (Bloqueo)
- **FIA_USB** - Vinculación de Sujeto a Usuario (Sesión)

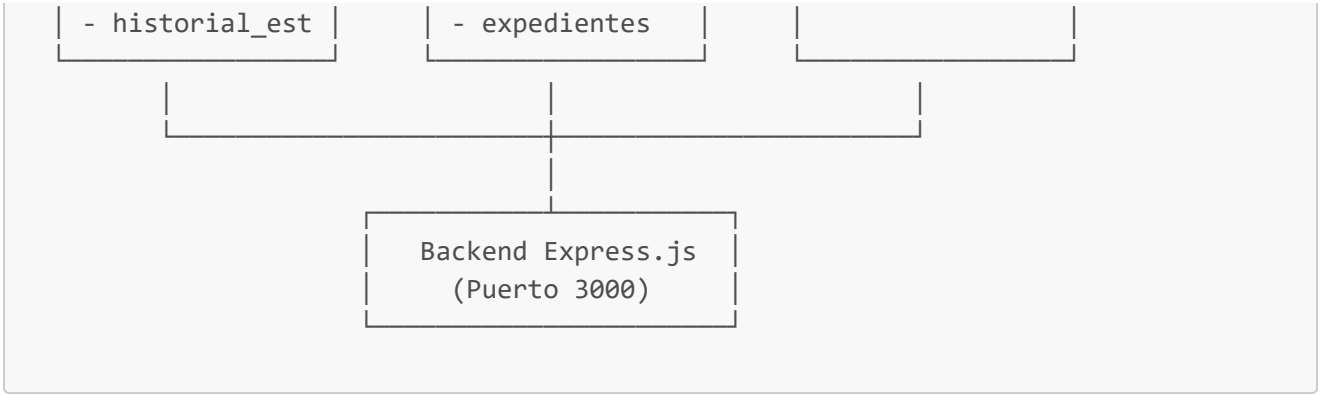
Características Principales Implementadas

1. **Dominio de correo institucional fijo**: `@judicatura.gob.ec`
2. **Verificación de disponibilidad en tiempo real** del usuario de correo
3. **Generación automática de contraseñas seguras** (12 caracteres)
4. **Envío de credenciales por correo electrónico** (nodemailer)
5. **Estado inicial HABILITABLE**: El admin debe activar manualmente la cuenta
6. **Gestión de estados de cuenta**: HABILITABLE, ACTIVA, SUSPENDIDA, BLOQUEADA, INACTIVA

🏗️ Arquitectura de Seguridad

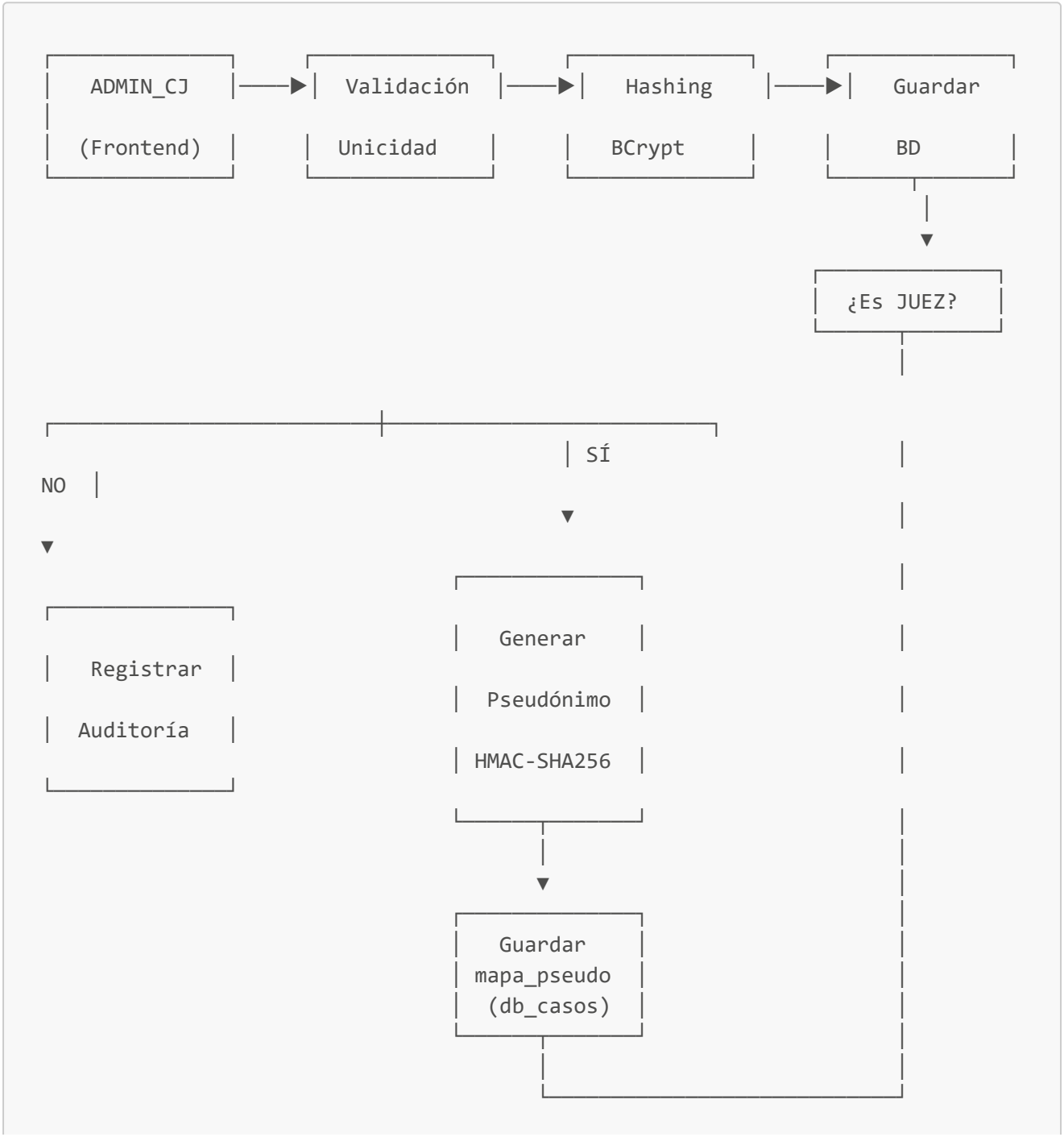
Bases de Datos Segregadas

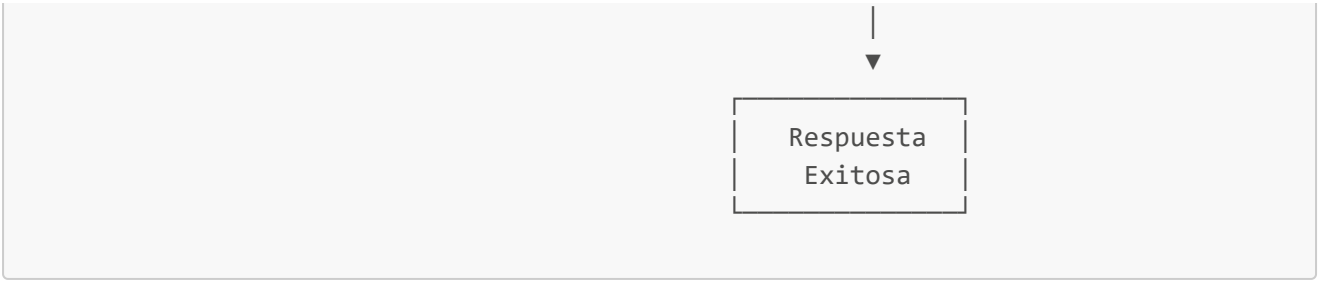




Flujo 1: Creación de Usuarios

Diagrama de Flujo





Implementación en Código

1. Validación de Unicidad

Archivo: backend/src/services/usuarios.service.ts

```
// Verificar si ya existe identificación o correo
const existe = await client.query(
  "SELECT funcionario_id FROM funcionarios WHERE identificacion = $1 OR
correo_institucional = $2",
  [input.identificacion, input.correoInstitucional.toLowerCase()]
);

if (existe.rows.length > 0) {
  throw new Error("Ya existe un funcionario con esa identificación o correo");
}
```

Seguridad aplicada:

- ☑ Previene duplicación de identidades
- ☑ Normaliza correo a minúsculas para consistencia
- ☑ Consulta parametrizada (prevención SQL Injection)

---

2. Hashing de Contraseña (BCrypt)

Archivo: backend/src/services/usuarios.service.ts

```
const passwordHash = await bcrypt.hash(input.password, 12);

const result = await client.query(
  `INSERT INTO funcionarios (
    identificacion, nombres_completos, correo_institucional, password_hash,
    rol_id, unidad_judicial, materia, estado, intentos_fallidos
  ) VALUES ($1, $2, $3, $4, $5, $6, $7, 'HABILITABLE', 0)
  RETURNING *`,
  [
    input.identificacion,
    input.nombresCompletos,
    input.correoInstitucional.toLowerCase(),
    passwordHash,
```

```
      input.rolId,
      input.unidadJudicial,
      input.materia,
    ]
  );
```

Seguridad aplicada:

- ☑ BCrypt con 12 rounds (factor de trabajo alto)
- ☑ Contraseña NUNCA se almacena en texto plano
- ☑ Estado inicial "HABILITABLE" (requiere activación)
- ☑ Intentos fallidos inician en 0

Configuración de BCrypt:

Archivo: `backend/src/config/index.ts`

```
security: {
  bcryptRounds: parseInt(process.env.BCRYPT_ROUNDS || "12", 10),
  maxLoginAttempts: parseInt(process.env.MAX_LOGIN_ATTEMPTS || "5", 10),
  lockoutDurationMinutes: parseInt(process.env.LOCKOUT_DURATION_MINUTES ||
"30", 10),
},
```

---

3. Generación de Pseudónimo para Jueces (HMAC-SHA256)

Archivo: `backend/src/services/pseudonimos.service.ts`

```
/**
 * Genera un pseudónimo único e irreversible para un juez
 * Usa HMAC-SHA256(JuezID + Salt + Timestamp) para máxima seguridad
 */
generatePseudonimo(juezId: number): string {
  // Crear datos para HMAC: ID + timestamp para unicidad adicional
  const data =
`${juezId}-${Date.now()}-${crypto.randomBytes(8).toString("hex")}`;

  // Generar HMAC-SHA256
  const hmac = crypto.createHmac("sha256", this.HMAC_SECRET);
  hmac.update(data);
  const hash = hmac.digest("hex");

  // Tomar los primeros 8 caracteres y convertir a formato amigable
  const shortCode = hash.substring(0, 8).toUpperCase();

  return `JUEZ-${shortCode}`;
}
```

Seguridad aplicada:

- ☑ HMAC-SHA256 (irreversible)
- ☑ Salt secreto configurable por variable de entorno
- ☑ Incluye timestamp y bytes aleatorios para unicidad
- ☑ Formato legible: `JUEZ-XXXXXXXX`

---

4. Integración: Crear Pseudónimo al Crear Juez

Archivo: `backend/src/services/usuarios.service.ts`

```
// EVENTO CRÍTICO: Si el rol es JUEZ, generar pseudónimo inmediatamente
// Esto garantiza que el pseudónimo exista ANTES de asignar cualquier causa
let pseudonimoGenerado: string | null = null;
if (input.rolId === ROL_JUEZ_ID) {
  pseudonimoGenerado = await pseudonimosService.crearPseudonimoJuez(
    funcionario.funcionario_id,
    adminId,
    ip,
    userAgent
  );
}
```

Almacenamiento del pseudónimo:

Archivo: `backend/src/services/pseudonimos.service.ts`

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

// Registrar en auditoría (sin revelar la relación ID-pseudónimo)
await auditService.log({
  tipoEvento: "CREACION_PSEUDONIMO",
  usuarioId: adminId,
  moduloAfectado: "ADMIN",
  descripcion: `Pseudónimo generado para nuevo juez`,
  datosAfectados: {
    // NO incluir juezId para mantener la privacidad
    pseudonimoGenerado: true
  },
  ipOrigen: ip,
```

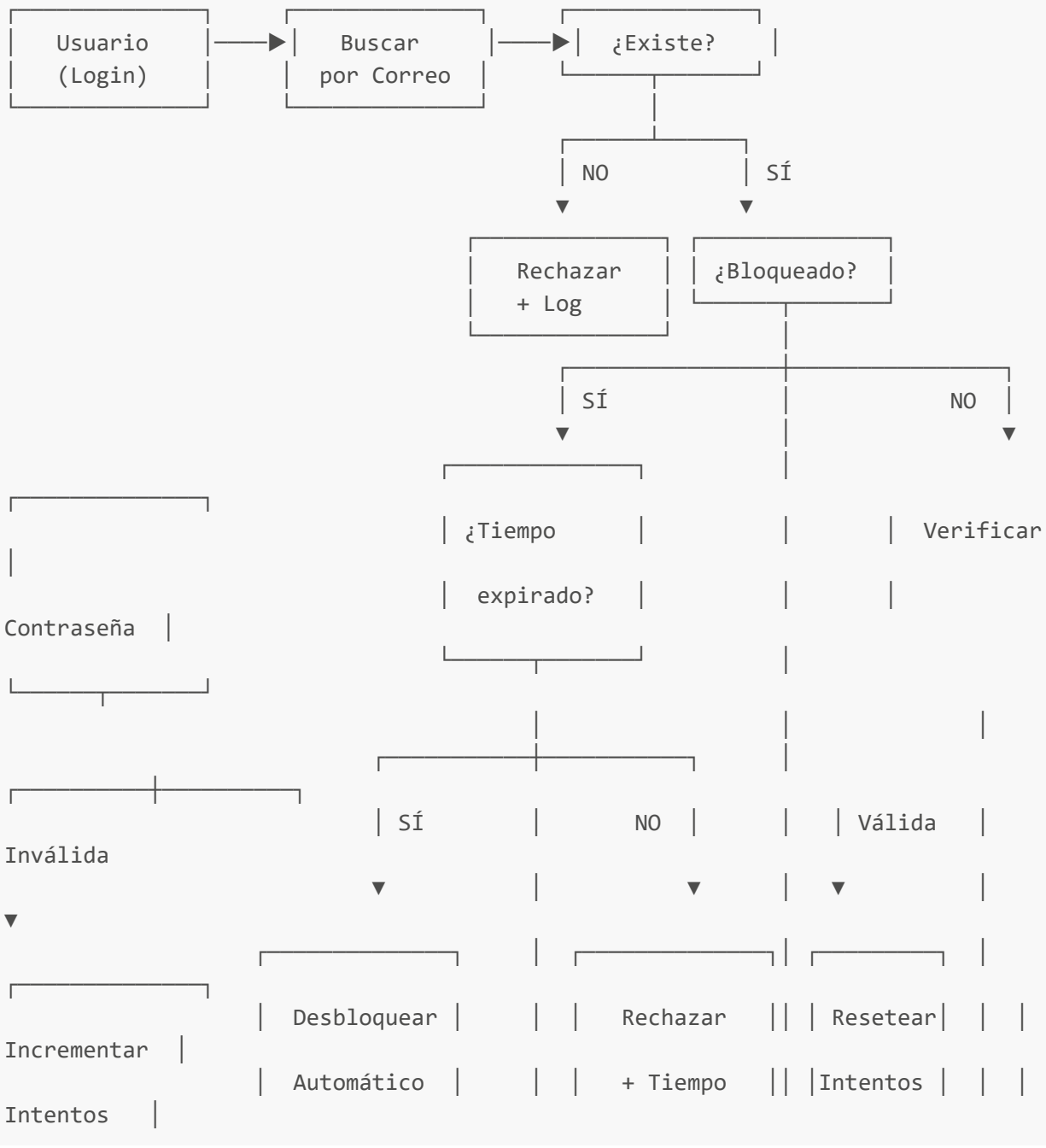
```
  userAgent,
});
```

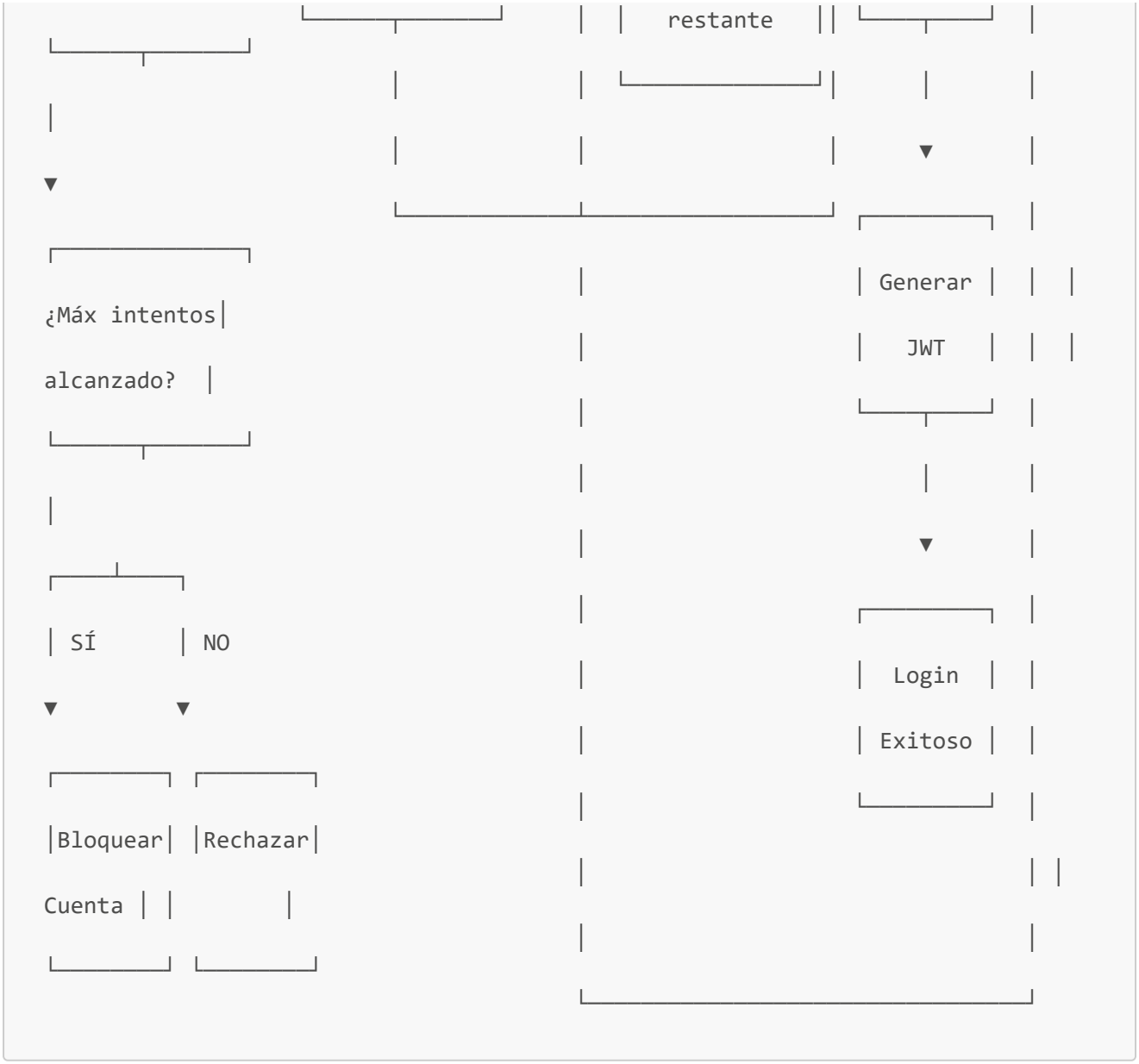
Seguridad aplicada:

- ☑ Pseudónimo se crea ANTES de cualquier asignación de causa
- ☑ Almacenado en base de datos separada (db_casos)
- ☑ Auditoría NO revela la relación ID-Pseudónimo
- ☑ Verificación de unicidad del pseudónimo generado

📊 Flujo 2: Autenticación (Login)

Diagrama de Flujo





Implementación en Código

1. Búsqueda de Usuario por Correo

Archivo: backend/src/services/auth.service.ts

```
// Buscar funcionario con su rol
const result = await client.query(
  `SELECT f.*, r.nombre as rol_nombre
   FROM funcionarios f
   JOIN roles r ON f.rol_id = r.rol_id
   WHERE f.correo_institucional = $1`,
  [correo.toLowerCase()]
);

const funcionario = result.rows[0] as (Funcionario & { rol_nombre: UserRole })
| undefined;
```

```
// Usuario no encontrado
if (!funcionario) {
  await auditService.logLogin(correo, ip, userAgent, false);
  return null;
}
```

Seguridad aplicada:

- ☑ Consulta parametrizada (SQL Injection prevention)
- ☑ Correo normalizado a minúsculas
- ☑ Log de intento fallido sin revelar si el usuario existe

---

2. Verificación de Bloqueo con Tiempo (FIA_AFL)

Archivo: backend/src/services/auth.service.ts

```
// Verificar si la cuenta está bloqueada (FIA_AFL)
if (funcionario.estado === "BLOQUEADA") {
  // Verificar si el tiempo de bloqueo ha expirado
  if (funcionario.fecha_bloqueo) {
    const tiempoBloqueoMs = config.security.lockoutDurationMinutes * 60 * 1000;
    const fechaDesbloqueo = new Date(funcionario.fecha_bloqueo.getTime() +
tiempoBloqueoMs);

    if (new Date() >= fechaDesbloqueo) {
      // Desbloquear automáticamente
      await client.query(
        `UPDATE funcionarios
         SET estado = 'ACTIVA', intentos_fallidos = 0, fecha_bloqueo = NULL,
fecha_actualizacion = NOW()
         WHERE funcionario_id = $1`,
        [funcionario.funcionario_id]
      );
      funcionario.estado = "ACTIVA" as EstadoCuenta;
      funcionario.intentos_fallidos = 0;
      funcionario.fecha_bloqueo = null;

      // Registrar desbloqueo automático
      await this.registrarCambioEstado(
        client,
        funcionario.funcionario_id,
        "BLOQUEADA",
        "ACTIVA",
        null // Sistema automático
      );
    } else {
      // Aún bloqueado
      const minutosRestantes = Math.ceil((fechaDesbloqueo.getTime() -
Date.now()) / 60000);
```

```
      await auditService.logLogin(correo, ip, userAgent, false,
funcionario.funcionario_id);
      throw new Error(`CUENTA_BLOQUEADA:${minutosRestantes}`);
    }
  } else {
    await auditService.logLogin(correo, ip, userAgent, false,
funcionario.funcionario_id);
    throw new Error("CUENTA_BLOQUEADA");
  }
}
```

Seguridad aplicada:

- ☑ Verificación de tiempo de bloqueo configurable
- ☑ Desbloqueo automático después de X minutos
- ☑ Registro en historial de cambios de estado
- ☑ Mensaje con tiempo restante para el usuario
- ☑ Log de cada intento en cuenta bloqueada

---

3. Verificación de Contraseña y Manejo de Fallos

Archivo: backend/src/services/auth.service.ts

```
// Verificar contraseña
const passwordValid = await bcrypt.compare(password,
funcionario.password_hash);

if (!passwordValid) {
  // Incrementar intentos fallidos (FIA_AFL)
  const nuevosIntentos = funcionario.intentos_fallidos + 1;
  let nuevoEstado: EstadoCuenta = funcionario.estado;
  let fechaBloqueo: Date | null = null;

  // Bloquear si excede intentos máximos
  if (nuevosIntentos >= config.security.maxLoginAttempts) {
    nuevoEstado = "BLOQUEADA" as EstadoCuenta;
    fechaBloqueo = new Date();

    // Registrar cambio de estado en historial
    await this.registrarCambioEstado(
      client,
      funcionario.funcionario_id,
      funcionario.estado,
      "BLOQUEADA",
      null // Sistema automático
    );
  }

  await client.query(
```

```
    `UPDATE funcionarios
     SET intentos_fallidos = $1, estado = $2, fecha_bloqueo = $3,
fecha_actualizacion = NOW()
     WHERE funcionario_id = $4`,
    [nuevosIntentos, nuevoEstado, fechaBloqueo, funcionario.funcionario_id]
  );

  await auditService.logLogin(correo, ip, userAgent, false,
funcionario.funcionario_id);
  return null;
}
```

Seguridad aplicada:

- ☑ BCrypt.compare para verificación segura
- ☑ Contador de intentos fallidos
- ☑ Bloqueo automático al alcanzar umbral (5 intentos por defecto)
- ☑ Registro de fecha de bloqueo
- ☑ Historial de cambios de estado
- ☑ Log de auditoría para cada intento

---

4. Login Exitoso y Generación de JWT (FIA_USB)

Archivo: backend/src/services/auth.service.ts

```
// Login exitoso - resetear intentos
await client.query(
  `UPDATE funcionarios
   SET intentos_fallidos = 0, fecha_bloqueo = NULL, fecha_actualizacion = NOW()
   WHERE funcionario_id = $1`,
  [funcionario.funcionario_id]
);

// Generar token JWT (FIA_USB - Atributos de sesión)
const payload: TokenPayload = {
  funcionarioId: funcionario.funcionario_id,
  identificacion: funcionario.identificacion,
  correo: funcionario.correo_institucional,
  rol: funcionario.rol_nombre,
  rolId: funcionario.rol_id,
  unidadJudicial: funcionario.unidad_judicial,
  materia: funcionario.materia,
};

const signOptions: SignOptions = {
  expiresIn: 1800, // 30 minutos
};

const token = jwt.sign(payload, config.jwt.secret as Secret, signOptions);
```

```
const expiresAt = new Date(Date.now() + 30 * 60 * 1000).toISOString();

await auditService.logLogin(correo, ip, userAgent, true,
funcionario.funcionario_id);

return {
  user: this.toPublicFuncionario(funcionario),
  token,
  expiresAt,
};
```

Seguridad aplicada:

- ☑ Reset de intentos fallidos en login exitoso
- ☑ JWT firmado con secreto configurable
- ☑ Expiración de 30 minutos
- ☑ Atributos de sesión incluidos (evita consultas constantes a BD):
  - `rol` - Para autorización
  - `unidadJudicial` - Para filtrado de datos
  - `materia` - Para filtrado de causas
- ☑ Log de auditoría de login exitoso

🔐 Estructura del Token JWT

Archivo: `backend/src/types/index.ts`

```
export interface TokenPayload {
  funcionarioId: number;
  identificacion: string;
  correo: string;
  rol: UserRole;           // "ADMIN_CJ" | "JUEZ" | "SECRETARIO"
  rolId: number;
  unidadJudicial: string;  // FIA_USB: Atributo de sesión
  materia: string;         // FIA_USB: Atributo de sesión
  iat?: number;            // Issued At (automático)
  exp?: number;            // Expiration (automático)
}
```

Ejemplo de Token Decodificado:

```
{
  "funcionarioId": 1,
  "identificacion": "1234567890",
  "correo": "juez.perez@funcionjudicial.gob.ec",
  "rol": "JUEZ",
  "rolId": 2,
```

```
  "unidadJudicial": "Unidad Judicial Civil Quito",
  "materia": "Civil",
  "iat": 1736000000,
  "exp": 1736001800
}
```

Registro de Auditoría (FAU)

Todas las acciones se registran en `db_logs.logs_auditoria` con hash SHA-256 para garantizar integridad:

Eventos de Autenticación

Evento	Descripción	Datos Registrados
`LOGIN_EXITOSO`	Usuario inicia sesión correctamente	usuario_id, correo, ip, user_agent
`LOGIN_FALLIDO`	Intento de login fallido	correo, ip, user_agent, usuario_id (si existe)
`LOGOUT`	Usuario cierra sesión	usuario_id, ip, user_agent
`CUENTA_BLOQUEADA`	Cuenta bloqueada por intentos fallidos	funcionario_id, intentos, fecha_bloqueo
`DESBLOQUEO_CUENTA`	Cuenta desbloqueada por admin	admin_id, funcionario_id, estado_anterior

Eventos de Gestión de Usuarios

Evento	Descripción	Datos Registrados
`CREACION_USUARIO`	Nuevo funcionario creado	admin_id, funcionario_id, identificacion, rol_id
`MODIFICACION_USUARIO`	Datos de funcionario actualizados	admin_id, funcionario_id, cambios
`CAMBIO_ESTADO`	Estado de cuenta modificado	admin_id, funcionario_id, estado_anterior, estado_nuevo
`CREACION_PSEUDONIMO`	Pseudónimo generado para juez	admin_id (NO se registra relación ID-pseudónimo)

Eventos de Consulta (Trazabilidad Completa)

Evento	Descripción	Datos Registrados
`CONSULTA_FUNCIONARIOS`	Admin consulta lista de funcionarios	admin_id, filtros, total_resultados

Evento	Descripción	Datos Registrados
CONSULTA_ROLES	Admin consulta roles disponibles	admin_id, total_roles
CONSULTA_JUECES	Consulta de jueces activos	usuario_id, total_jueces
CONSULTA_FUNCIONARIO	Consulta de funcionario específico	admin_id, funcionario_id_consultado
CONSULTA_FUNCIONARIO_NO_ENCONTRADO	Intento de consulta a ID inexistente	admin_id, funcionario_id_buscado
CONSULTA_HISTORIAL_ESTADOS	Consulta historial de estados	admin_id, funcionario_id, registros_historial
CONSULTA_AUDITORIA	Consulta de logs de auditoría	admin_id, filtros, registros

Implementación de Auditoría

Archivo: backend/src/services/audit.service.ts

```
/**
 * Registra un evento de auditoría con hash de integridad
 */
async log(event: LogEventInput): Promise<number> {
  const client = await logsPool.connect();

  try {
    const fechaEvento = new Date();

    // Crear hash del evento para integridad (SHA-256)
    const hashData = JSON.stringify({
      ...event,
      fechaEvento: fechaEvento.toISOString(),
      timestamp: Date.now(),
    });
    const hashEvento =
crypto.createHash("sha256").update(hashData).digest("hex");

    // Insertar en la base de datos
    const result = await client.query(
      `INSERT INTO logs_auditoria (
        fecha_evento, usuario_id, rol_usuario, ip_origen,
        tipo_evento, modulo_afectado, descripcion_evento, datos_afectados,
hash_evento
      ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)
      RETURNING log_id`,
      [
        fechaEvento,
```

```
        event.usuarioId,
        event.rolUsuario || null,
        event.ipOrigen,
        event.tipoEvento,
        event.moduloAfectado || null,
        event.descripcion || null,
        datos ? JSON.stringify(datos) : null,
        hashEvento,  // Hash SHA-256 para verificar integridad
      ]
    );

    return result.rows[0].log_id;
  } finally {
    client.release();
  }
}
```

Ejemplo de Registro en Rutas

Archivo: `backend/src/routes/usuarios.routes.ts`

```
// Registrar consulta en auditoría
await auditService.log({
  tipoEvento: "CONSULTA_FUNCIONARIO",
  usuarioId: req.user!.funcionarioId,
  moduloAfectado: "ADMIN",
  descripcion: `Consulta de funcionario: ${funcionario.identificacion}`,
  datosAfectados: {
    funcionarioIdConsultado: id,
    identificacion: funcionario.identificacion
  },
  ipOrigen: getClientIp(req),
  userAgent: getUserAgent(req),
});
```

Seguridad aplicada:

- ☑ Hash SHA-256 en cada registro (inmutabilidad)
- ☑ IP de origen para trazabilidad geográfica
- ☑ User-Agent para identificar dispositivo
- ☑ Timestamp preciso de cada evento
- ☑ Base de datos separada (db_logs) para aislamiento
- ☑ Auditoría de TODAS las consultas, no solo modificaciones

⚙ Configuración de Seguridad

Archivo: `backend/.env.example`

```
# Seguridad
BCRYPT_ROUNDS=12                    # Factor de trabajo para hashing
MAX_LOGIN_ATTEMPTS=5                # Intentos antes de bloqueo
LOCKOUT_DURATION_MINUTES=30         # Duración del bloqueo

# Pseudónimos de Jueces (FDP - Protección de Identidad)
PSEUDONIMO_HMAC_SECRET=your-hmac-secret-change-in-production

# JWT
JWT_SECRET=your-super-secret-jwt-key-change-in-production
JWT_EXPIRES_IN=30m
```

Archivos Involucrados

Archivo	Responsabilidad
`backend/src/services/auth.service.ts`	Login, verificación, JWT
`backend/src/services/usuarios.service.ts`	CRUD de funcionarios
`backend/src/services/pseudonimos.service.ts`	Generación HMAC-SHA256
`backend/src/services/audit.service.ts`	Registro de auditoría
`backend/src/types/index.ts`	Definición de tipos
`backend/src/config/index.ts`	Configuración centralizada
`backend/src/middleware/auth.middleware.ts`	Validación de JWT

Cumplimiento Common Criteria

Requisito	Estado	Implementación
FIA_UID.1	☑	Identificación por correo institucional
FIA_UAU.1	☑	Autenticación por contraseña hasheada (BCrypt 12 rounds)
FIA_ATD.1	☑	Atributos: rol, unidad judicial, materia
FIA_AFL.1	☑	Bloqueo después de 5 intentos fallidos, desbloqueo automático
FIA_USB.1	☑	Atributos en JWT firmado para sesión
FDP_IFF.1	☑	Pseudónimos irreversibles HMAC-SHA256 para jueces
FAU_GEN.1	☑	Registro de TODOS los eventos (CRUD + consultas)
FAU_GEN.2	☑	Identificación de usuario en cada evento
FAU_STG.1	☑	Almacenamiento en BD separada (db_logs)

Requisito	Estado	Implementación
FAU_STG.4	☑	Hash SHA-256 para integridad de logs

🩹 Casos de Prueba Recomendados

Autenticación

#	Caso	Entrada	Resultado Esperado
1	Login exitoso	Correo y contraseña válidos	JWT + log `LOGIN_EXITOSO`
2	Login fallido	Contraseña incorrecta	null + log `LOGIN_FALLIDO` + incremento intentos
3	Bloqueo automático	5 contraseñas incorrectas	Error `CUENTA_BLOQUEADA` + log
4	Login bloqueado	Cuenta bloqueada < 30 min	Error con minutos restantes
5	Desbloqueo automático	Cuenta bloqueada > 30 min	Login permitido + log desbloqueo

Gestión de Usuarios

#	Caso	Entrada	Resultado Esperado
6	Crear usuario duplicado	Identificación existente	Error 409 + NO log de creación
7	Crear JUEZ	Datos válidos + rol=2	Usuario + pseudónimo + 2 logs
8	Crear SECRETARIO	Datos válidos + rol=3	Usuario + 1 log (sin pseudónimo)
9	Actualizar funcionario	Cambios válidos	Usuario actualizado + log
10	Cambiar estado	ACTIVA → SUSPENDIDA	Estado cambiado + log + historial

Trazabilidad de Consultas

#	Caso	Acción	Log Esperado
11	Consultar lista	GET /api/usuarios	`CONSULTA_FUNCIONARIOS`
12	Consultar roles	GET /api/usuarios/roles	`CONSULTA_ROLES`
13	Consultar jueces	GET /api/usuarios/jueces	`CONSULTA_JUECES`
14	Consultar uno	GET /api/usuarios/5	`CONSULTA_FUNCIONARIO`
15	Consultar inexistente	GET /api/usuarios/999	`CONSULTA_FUNCIONARIO_NO_ENCONTRADO`

#	Caso	Acción	Log Esperado
16	Consultar historial	GET /api/usuarios/5/historial	`CONSULTA_HISTORIAL_ESTADOS`

Documento generado para el proyecto Juez Seguro - Desarrollo de Software Seguro - EPN 2026