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:

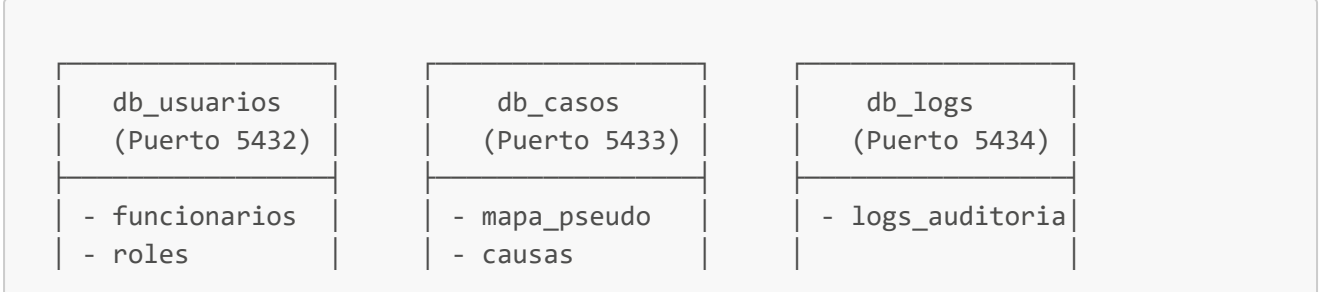
- **FIA_UID** - Identificación de Usuario
- **FIA_ATD** - Definición de Atributos de Usuario
- **FIA_AFL** - Manejo de Fallos de Autenticación (Bloqueo)
- **FIA_USB** - Vinculación de Sujeto a Usuario (Sesión)

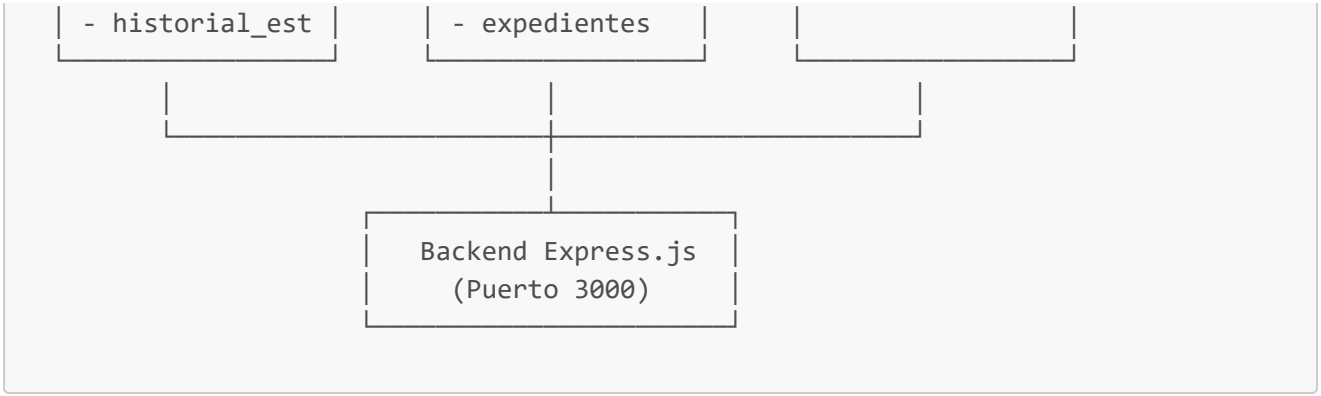
Características Principales Implementadas

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

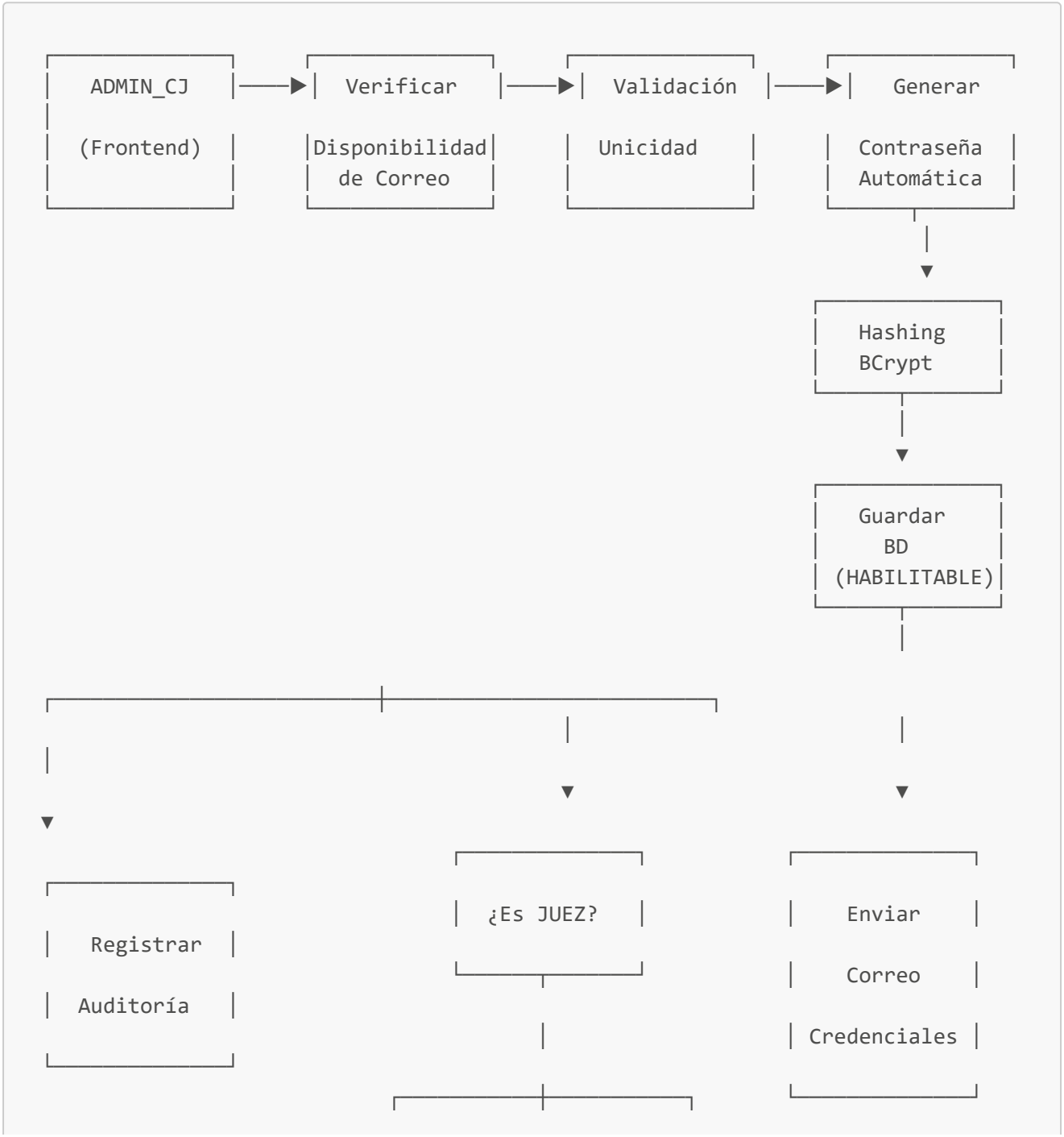
Bases de Datos Segregadas

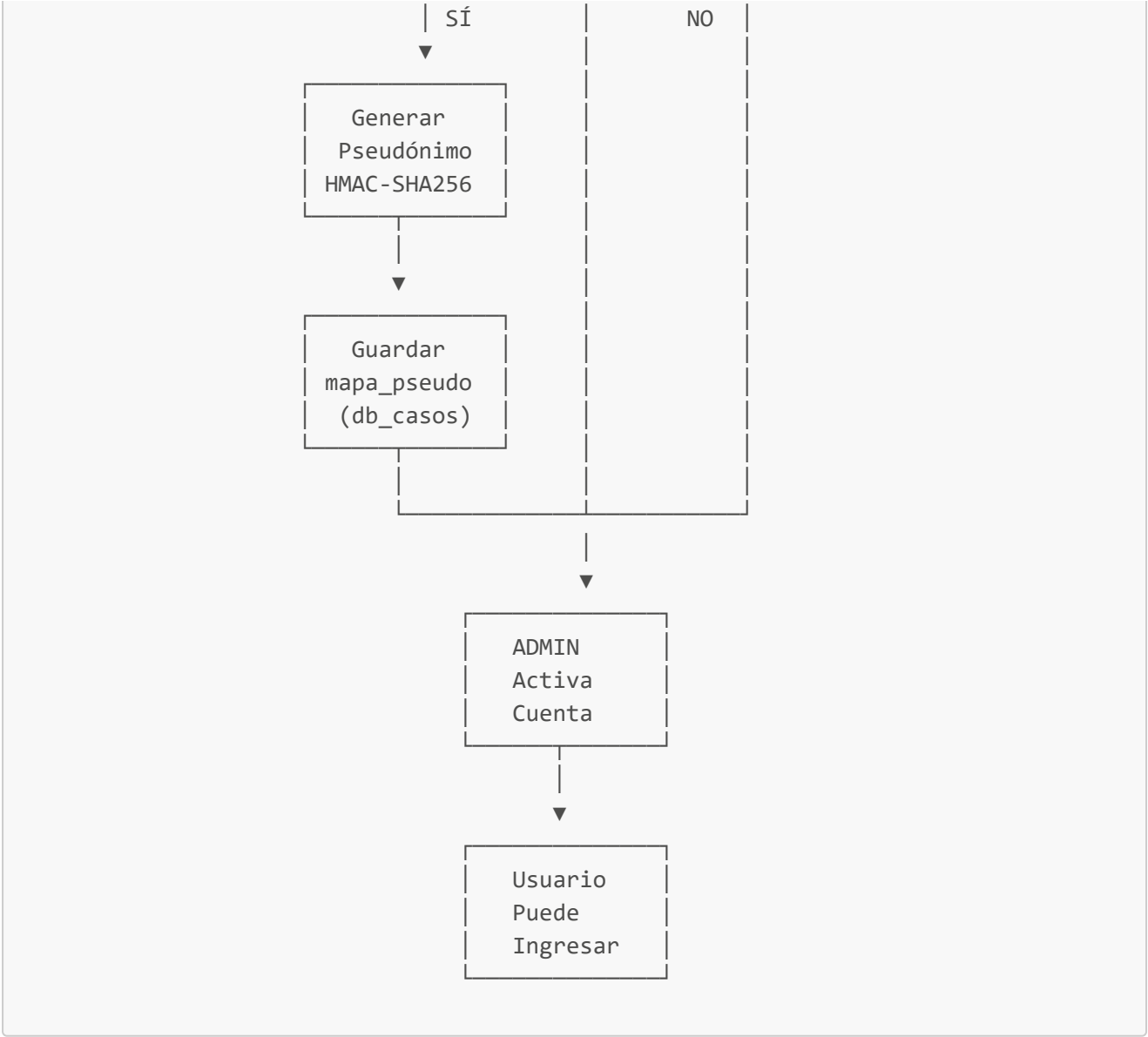




📊 Flujo 1: Creación de Usuarios

Diagrama de Flujo





Implementación en Código

1. Verificación de Disponibilidad de Correo (Tiempo Real)

Archivo: frontend/src/pages/funcionarios/GestionCuentas.tsx

```
// Dominio institucional fijo
const DOMINIO_INSTITUCIONAL = "@judicatura.gob.ec";

// Función para verificar disponibilidad del correo con debounce
const verificarDisponibilidadEmail = useCallback(async (prefix: string) => {
  if (prefix.length < 3) {
    setEmailDisponibilidad("idle");
    return;
  }

  setEmailDisponibilidad("checking");

  try {
```

```
    const correoCompleto = `${prefix.toLowerCase()}${DOMINIO_INSTITUCIONAL}`;
    const resultado = await
usuariosService.verificarDisponibilidad(correoCompleto);
    setEmailDisponibilidad(resultado.disponible ? "available" : "taken");
  } catch {
    setEmailDisponibilidad("idle");
  }
}, []);
```

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

```
// GET /api/usuarios/verificar-disponibilidad
router.get(
  "/verificar-disponibilidad",
  authenticate,
  authorize("ADMIN_CJ"),
  async (req: Request, res: Response, next: NextFunction) => {
    const correo = req.query.correo as string;
    const disponible = await
funcionariosService.verificarDisponibilidadCorreo(correo.toLowerCase());
    res.json({ success: true, data: { disponible } });
  }
);
```

Seguridad aplicada:

- ☑ Verificación en tiempo real mientras el admin escribe
- ☑ Debounce de 500ms para evitar sobrecarga
- ☑ Dominio fijo @judicatura.gob.ec (no editable)
- ☑ Solo caracteres permitidos: a-z, 0-9, ., -, _

---

2. 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)

---

3. Generación Automática de Contraseña Segura

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

```
/**
 * Genera una contraseña segura aleatoria
 * Incluye mayúsculas, minúsculas, números y caracteres especiales
 */
function generarPasswordSeguro(longitud: number = 12): string {
  const mayusculas = "ABCDEFGHJKLMNPQRSTUVWXYZ"; // Sin I, O para evitar
confusión
  const minusculas = "abcdefghjkmnpqrstuvwxyz"; // Sin i, l, o
  const numeros = "23456789"; // Sin 0, 1 para evitar confusión
  const especiales = "!@#$%&*";

  // Garantizar al menos uno de cada tipo
  let password = "";
  password += mayusculas[crypto.randomInt(mayusculas.length)];
  password += minusculas[crypto.randomInt(minusculas.length)];
  password += numeros[crypto.randomInt(numeros.length)];
  password += especiales[crypto.randomInt(especiales.length)];

  // Completar con caracteres aleatorios
  const todosCaracteres = mayusculas + minusculas + numeros + especiales;
  for (let i = password.length; i < longitud; i++) {
    password += todosCaracteres[crypto.randomInt(todosCaracteres.length)];
  }

  // Mezclar los caracteres
  return password.split("").sort(() => crypto.randomInt(3) - 1).join("");
}
```

Seguridad aplicada:

- ☑ Contraseña generada en el servidor (nunca en frontend)
- ☑ Mínimo 12 caracteres
- ☑ Incluye mayúsculas, minúsculas, números y símbolos
- ☑ Caracteres ambiguos eliminados (0, O, 1, l, I)
- ☑ Uso de `crypto.randomInt()` (criptográficamente seguro)

---

4. Hashing de Contraseña (BCrypt)

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

```
// Generar contraseña automáticamente si no se proporciona
const passwordTemporal = input.password || generarPasswordSeguro(12);
const passwordHash = await bcrypt.hash(passwordTemporal, 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 manual)
- ☑ Intentos fallidos inician en 0

---

5. Envío de Credenciales por Correo Electrónico

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

```
/**
 * Envía las credenciales de acceso a un nuevo funcionario
 */
async enviarCredenciales(
  correo: string,
  nombreCompleto: string,
  passwordTemporal: string
): Promise<boolean> {
  const loginUrl = `${config.frontendUrl}/funcionarios/login`;

  const htmlContent = `
    <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
auto;">
      <div style="background: linear-gradient(135deg, #1e3a5f 0%, #2d5a87
```

```
100%); padding: 30px; text-align: center;">
        <h1 style="color: white; margin: 0;">⚖️ Sistema Juez Seguro</h1>
        <p style="color: #e0e0e0; margin-top: 10px;">Consejo de la
Judicatura</p>
      </div>

      <div style="padding: 30px; background: #f9f9f9;">
        <h2 style="color: #1e3a5f;">Bienvenido/a al Sistema</h2>
        <p>Estimado/a <strong>${nombreCompleto}</strong>,</p>
        <p>Se ha creado su cuenta en el Sistema Juez Seguro.</p>

        <div style="background: white; border-radius: 8px; padding: 20px;
margin: 20px 0;">
          <p><strong>📧 Correo:</strong> ${correo}</p>
          <p><strong>🔐 Contraseña temporal:</strong>
<code>${passwordTemporal}</code></p>
        </div>

        <div style="background: #fff3cd; border: 1px solid #ffc107; padding:
15px; border-radius: 8px;">
          <p style="margin: 0; color: #856404;">
            ⚠️ <strong>Importante:</strong> Su cuenta debe ser activada por el
administrador
            antes de poder iniciar sesión. Será notificado cuando esto ocurra.
          </p>
        </div>
      </div>
    </div>
  `;

  // En modo desarrollo, solo loguea el correo
  if (!config.email.user || !config.email.pass) {
    console.log("📧 CORREO (modo desarrollo)");
    console.log(`Para: ${correo}`);
    console.log(`Contraseña: ${passwordTemporal}`);
    return true;
  }

  // Enviar correo real
  await this.transporter.sendMail({
    from: `"${config.email.fromName}" <${config.email.from}>`,
    to: correo,
    subject: "🔐 Credenciales de Acceso - Sistema Juez Seguro",
    html: htmlContent,
  });

  return true;
}
```

Seguridad aplicada:

- ☑ Contraseña enviada solo por correo institucional

- ☑ Modo desarrollo: credenciales en consola (no se envía correo real)
- ☑ Modo producción: SMTP configurable (Gmail, Exchange, etc.)
- ☑ Notificación clara sobre activación pendiente

---

6. Estados de Cuenta y Activación Manual

Archivo: `frontend/src/pages/funcionarios/GestionCuentas.tsx`

```
// Estados de cuenta disponibles
type EstadoCuenta = "habilitable" | "activa" | "suspendida" | "bloqueada" |
"inactiva";

// Badges visuales para cada estado
const getEstadoBadge = (estado: EstadoCuenta) => {
  switch (estado) {
    case "activa":
      return <Badge className="bg-success/10 text-success">Activa</Badge>;
    case "habilitable":
      return <Badge className="bg-blue-500/10 text-blue-600">Pendiente
Activación</Badge>;
    case "suspendida":
      return <Badge className="bg-warning/10 text-warning">Suspendida</Badge>;
    case "bloqueada":
      return <Badge className="bg-destructive/10 text-
destructive">Bloqueada</Badge>;
    case "inactiva":
      return <Badge className="bg-muted text-muted-
foreground">Inactiva</Badge>;
  }
};

// Acciones de cambio de estado
{funcionario.estado !== "activa" && (
  <DropdownMenuItem onClick={() => handleStatusChange(funcionario.id,
"activa")}>
    <UserCheck className="w-4 h-4 mr-2 text-success" />
    Activar
  </DropdownMenuItem>
)}
```

Flujo de activación:

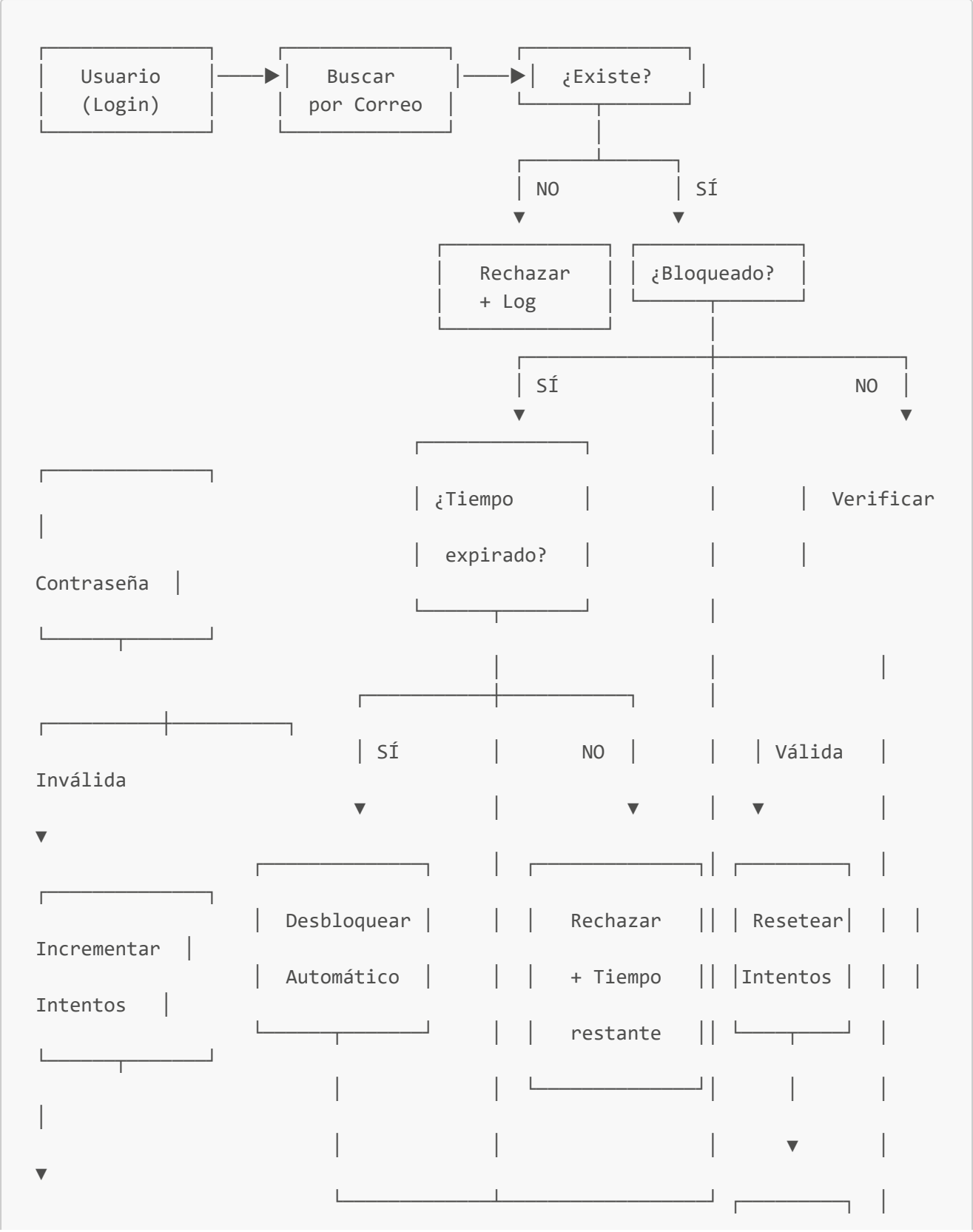
1. Admin crea funcionario → Estado: `HABILITABLE`
2. Se envía correo con credenciales
3. Admin revisa y activa manualmente → Estado: `ACTIVA`
4. Funcionario puede iniciar sesión

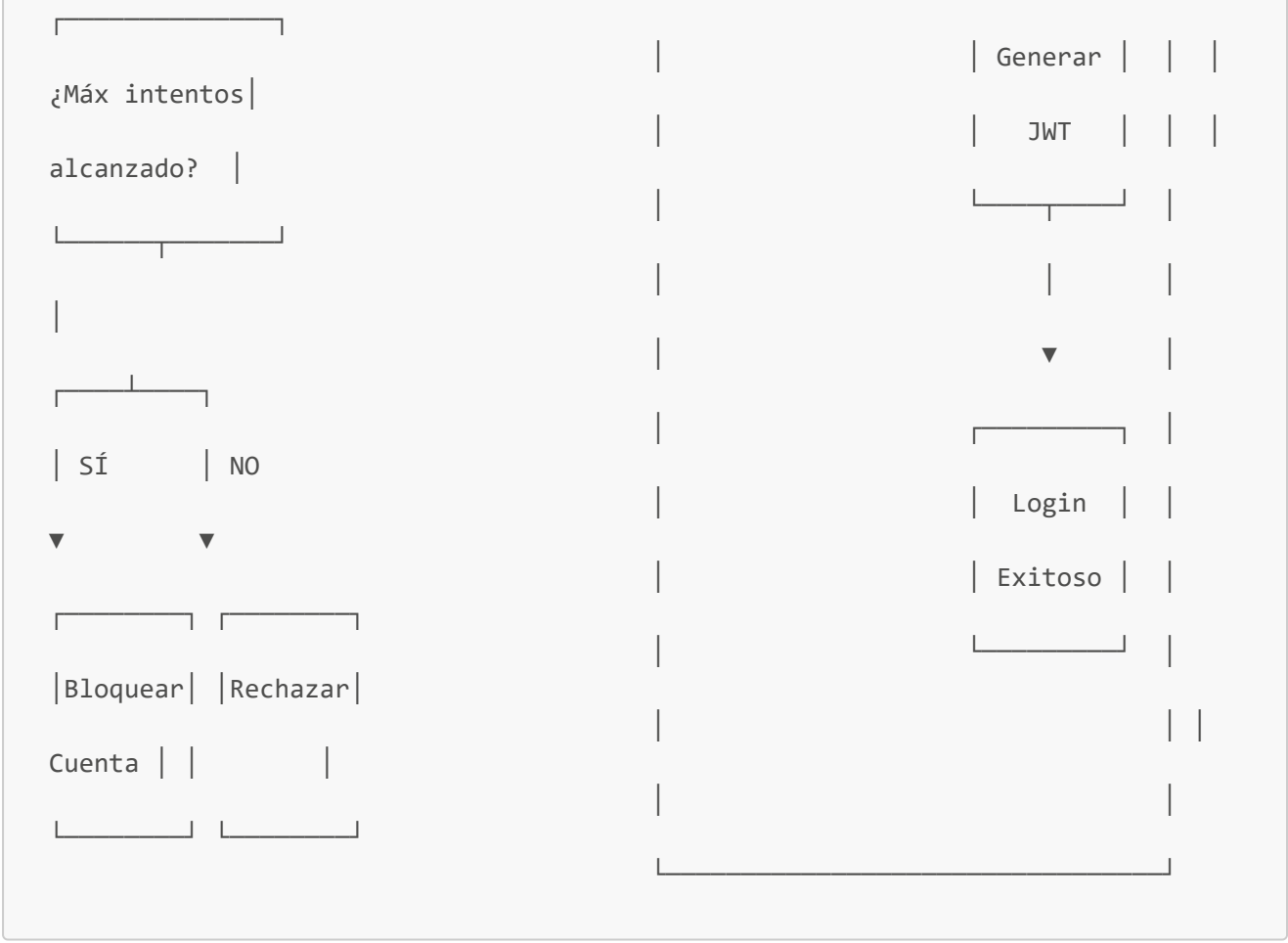
Seguridad aplicada:

- ☑ Doble verificación: creación + activación son pasos separados
- ☑ Admin tiene control total sobre quién puede acceder
- ☑ Previene accesos no autorizados por errores de creación

📊 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`

```typescript
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:

```json
{
  "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
`CONSULTA_ROLES`	Admin consulta roles disponibles	admin_id, total_roles
`CONSULTA_JUECES`	Consulta de jueces activos	usuario_id, total_jueces

Evento	Descripción	Datos Registrados
`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

# Correo Electrónico (SMTP)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=tu-correo@gmail.com
SMTP_PASS=tu-app-password
SMTP_FROM=noreply@judicatura.gob.ec
SMTP_FROM_NAME=Sistema Juez Seguro
FRONTEND_URL=http://localhost:8080
```

Notas sobre configuración de correo:

- En desarrollo: Las credenciales se muestran en consola (no se envían correos reales)
- En producción: Configurar variables SMTP para envío real
- Para Gmail: Usar "Contraseña de aplicación" (no la contraseña normal)

Archivos Involucrados

Backend

Archivo	Responsabilidad
`backend/src/services/auth.service.ts`	Login, verificación, JWT
`backend/src/services/usuarios.service.ts`	CRUD de funcionarios, generación de contraseñas
`backend/src/services/email.service.ts`	Envío de correos con credenciales
`backend/src/services/pseudonimos.service.ts`	Generación HMAC-SHA256
`backend/src/services/audit.service.ts`	Registro de auditoría
`backend/src/routes/usuarios.routes.ts`	Endpoints de gestión de usuarios
`backend/src/types/index.ts`	Definición de tipos
`backend/src/config/index.ts`	Configuración centralizada (incluye SMTP)
`backend/src/middleware/auth.middleware.ts`	Validación de JWT

Frontend

Archivo	Responsabilidad
`frontend/src/pages/funcionarios/GestionCuentas.tsx`	Formulario de creación/edición
`frontend/src/pages/funcionarios/LoginFuncionarios.tsx`	Página de login
`frontend/src/services/usuarios.service.ts`	Cliente API de usuarios
`frontend/src/contexts/AuthContext.tsx`	Estado de autenticación
`frontend/src/components/funcionarios/FuncionariosSidebar.tsx`	Navegación y logout
`frontend/src/lib/funcionarios-data.ts`	Mapeo de datos backend→frontend
`frontend/src/types/index.ts`	Tipos del frontend

☑ 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)
FAU_STG.4	☑	Hash SHA-256 para integridad de logs

🧪 Casos de Prueba Recomendados

Creación de Usuarios

#	Caso	Entrada	Resultado Esperado
1	Verificar disponibilidad correo libre	`juan.perez`	Indicador verde ☑ "Usuario disponible"
2	Verificar disponibilidad correo ocupado	`admin.cj`	Indicador rojo ✗ "Este usuario ya está en uso"

#	Caso	Entrada	Resultado Esperado
3	Crear usuario con correo disponible	Datos válidos	Usuario creado + estado HABILITABLE + correo enviado
4	Crear usuario duplicado	Identificación existente	Error 409 + mensaje "Ya existe un funcionario..."
5	Crear JUEZ	Datos válidos + rol=2	Usuario + pseudónimo + correo + 2 logs
6	Crear SECRETARIO	Datos válidos + rol=3	Usuario + correo + 1 log (sin pseudónimo)
7	Validar formato correo	`abc`	Error "El usuario debe tener al menos 3 caracteres"
8	Caracteres inválidos	`Juan.Pérez`	Error "Solo letras minúsculas, números, puntos..."

Activación de Cuentas

#	Caso	Acción	Resultado Esperado
9	Activar cuenta HABILITABLE	Acción "Activar" en dropdown	Estado → ACTIVA + usuario puede hacer login
10	Login con cuenta HABILITABLE	Intentar login	Error "Cuenta no activa. Contacte al administrador"
11	Suspender cuenta activa	Acción "Suspender"	Estado → SUSPENDIDA + login rechazado
12	Reactivar cuenta suspendida	Acción "Activar"	Estado → ACTIVA + login permitido

Autenticación

#	Caso	Entrada	Resultado Esperado
13	Login exitoso	Correo y contraseña válidos	JWT + log `LOGIN_EXITOSO`
14	Login fallido	Contraseña incorrecta	null + log `LOGIN_FALLIDO` + incremento intentos
15	Bloqueo automático	5 contraseñas incorrectas	Error `CUENTA_BLOQUEADA` + log
16	Login bloqueado	Cuenta bloqueada < 30 min	Error con minutos restantes

#	Caso	Entrada	Resultado Esperado
17	Desbloqueo automático	Cuenta bloqueada > 30 min	Login permitido + log desbloqueo

Gestión de Sesión

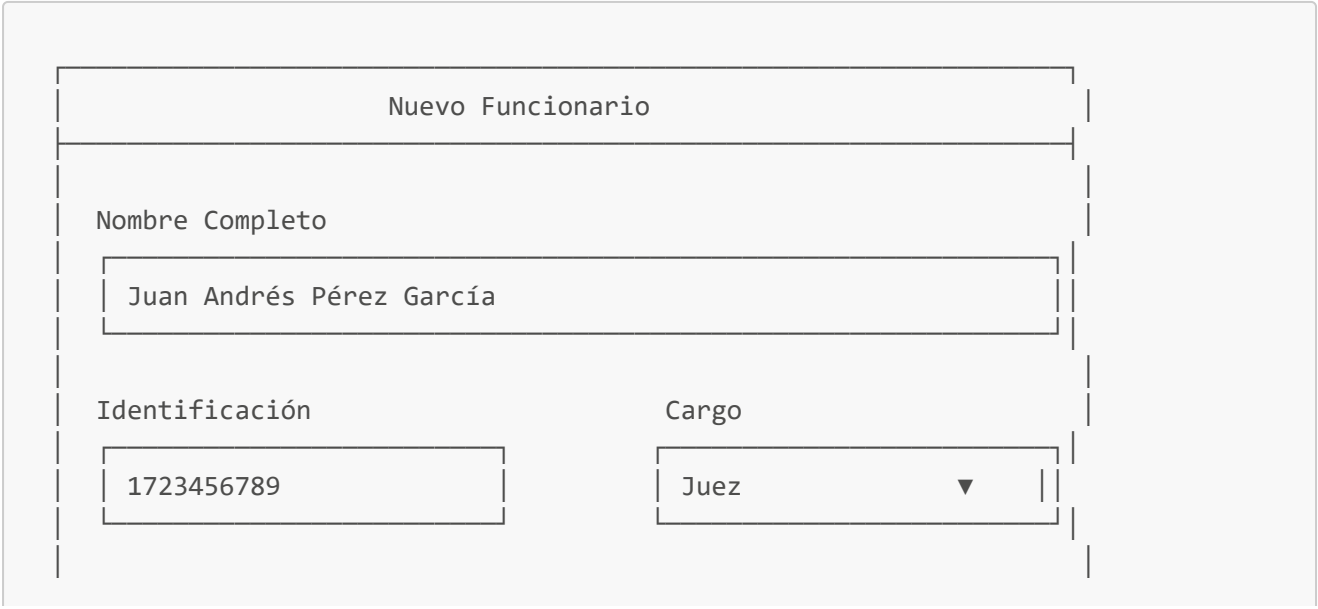
#	Caso	Acción	Resultado Esperado
18	Cerrar sesión	Click en "Cerrar Sesión"	Redirigir a login + limpiar token
19	Login nuevo usuario	Login después de logout	Redirigir a dashboard (no a ruta anterior)
20	Sesión expirada	Token > 30 min	Redirigir a login al intentar acción

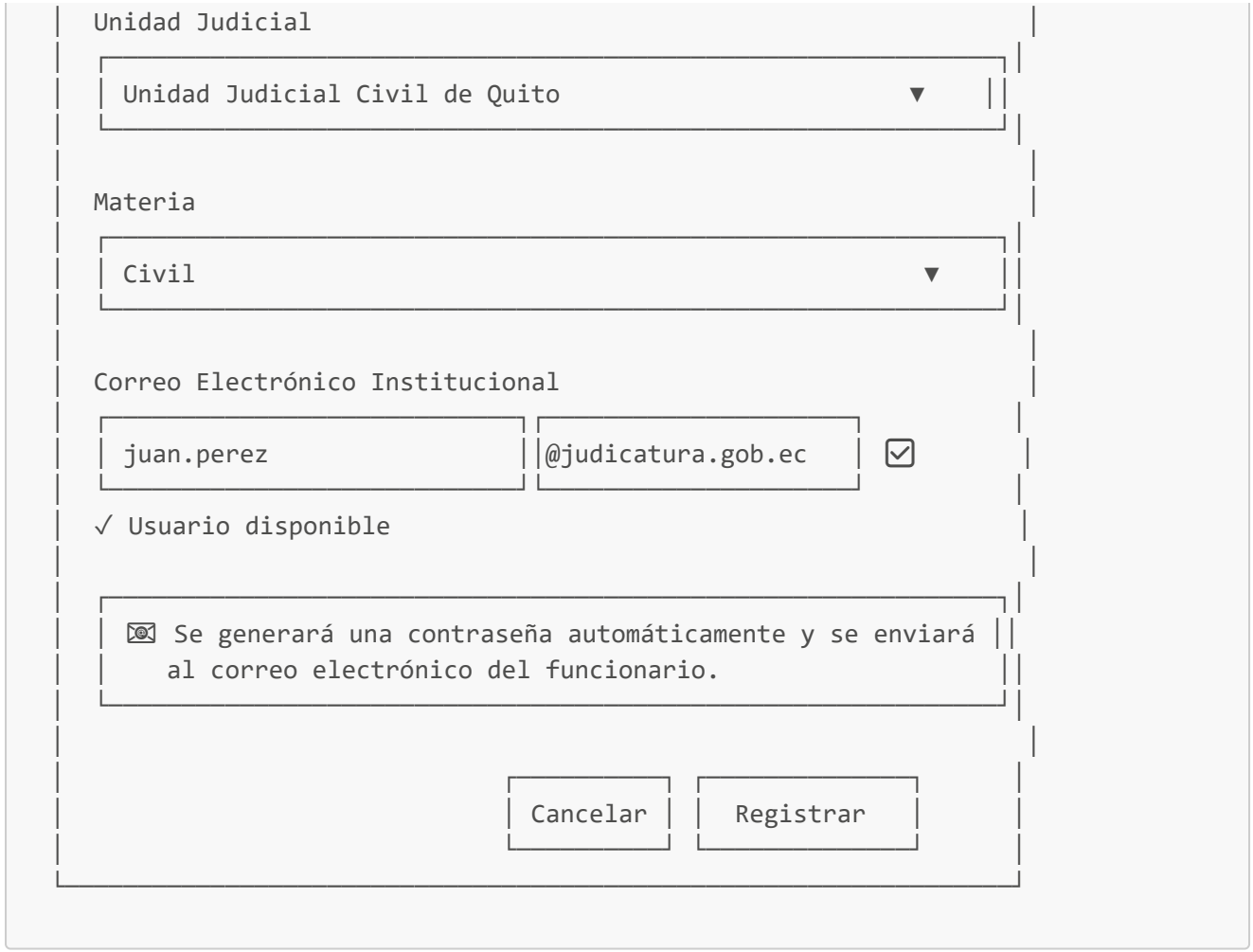
Trazabilidad de Consultas

#	Caso	Acción	Log Esperado
21	Consultar lista	GET /api/usuarios	`CONSULTA_FUNCIONARIOS`
22	Consultar roles	GET /api/usuarios/roles	`CONSULTA_ROLES`
23	Consultar jueces	GET /api/usuarios/jueces	`CONSULTA_JUECES`
24	Consultar uno	GET /api/usuarios/5	`CONSULTA_FUNCIONARIO`
25	Consultar inexistente	GET /api/usuarios/999	`CONSULTA_FUNCIONARIO_NO_ENCONTRADO`
26	Verificar disponibilidad	GET /api/usuarios/verificar-disponibilidad	(Sin log - operación frecuente)

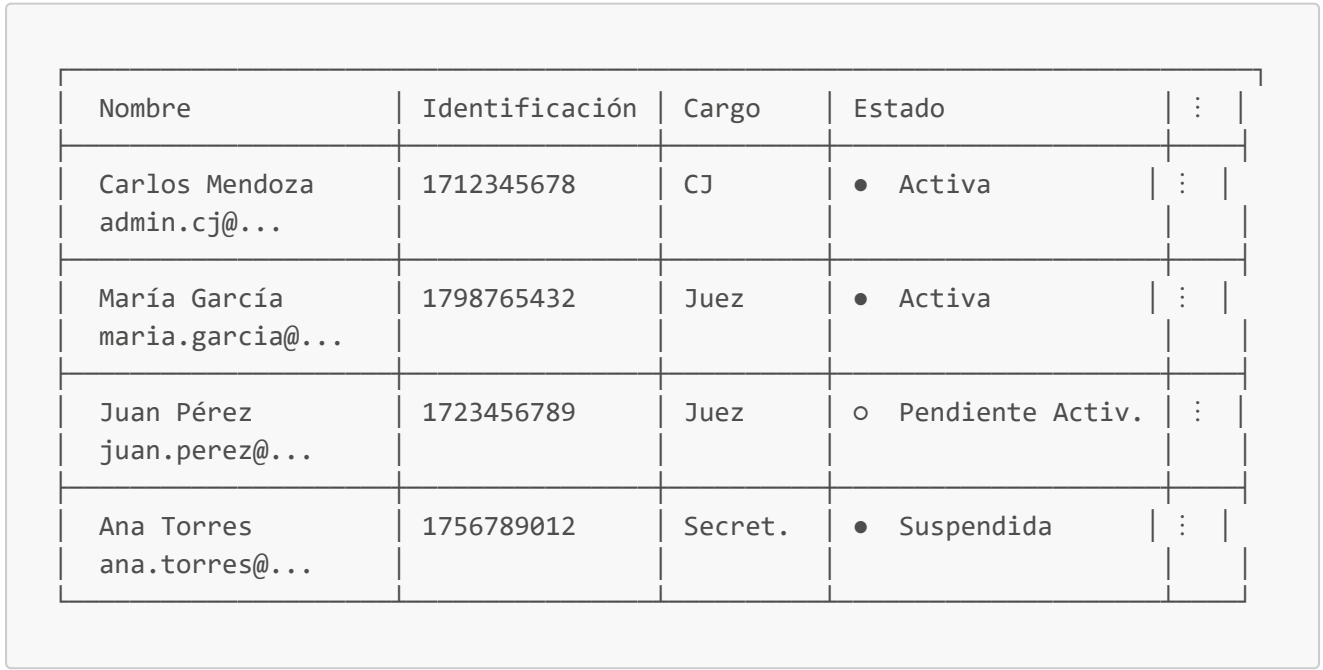
📸 Capturas de Pantalla del Flujo

Formulario de Creación de Funcionario





Lista de Funcionarios con Estados



Documento actualizado el 5 de enero de 2026
Proyecto Juez Seguro - Desarrollo de Software Seguro - EPN 2026