Table of Contents

# Manual Técnico: VecinosExpress

## 1. Arquitectura del Sistema

### Arquitectura General

┌───────────────────┐ ┌───────────────────┐ ┌───────────────────┐  
│ Cliente - React │<────│ Servidor - Node │<────│ PostgreSQL DB │  
│ UI/UX Components │ │ Express API │ │ Drizzle ORM │  
└───────────────────┘ └───────────────────┘ └───────────────────┘  
 ▲ ▲ ▲  
 │ │ │  
 ▼ ▼ ▼  
┌───────────────────┐ ┌───────────────────┐ ┌───────────────────┐  
│ Capa Servicios │ │ Servicios Externos│ │ Seguridad │  
│ NFC, Signatures │ │ Agora, Payments │ │ Auth, Encryption │  
└───────────────────┘ └───────────────────┘ └───────────────────┘

### Componentes Principales

1. **Frontend (React/TypeScript)**:
   * Interfaz de usuario responsive
   * Gestión de estados con React Hooks
   * Shadcn UI para componentes visuales
   * TailwindCSS para estilos
2. **Backend (Node.js/Express)**:
   * API RESTful para todas las operaciones
   * Autenticación y autorización
   * Integración con servicios externos
   * Lógica de negocio modular
3. **Base de Datos (PostgreSQL)**:
   * Drizzle ORM para consultas y migraciones
   * Esquema relacional optimizado
   * Índices para consultas frecuentes
4. **Integraciones Externas**:
   * Pasarelas de pago (MercadoPago, Stripe, PayPal)
   * Servicio de videoconferencia (Agora)
   * Sistema de correo electrónico (SendGrid)
   * Verificación de identidad (GetAPI.cl)

## 2. Estructura de Carpetas

├── client/ # Aplicación frontend React  
│ ├── src/ # Código fuente frontend  
│ │ ├── components/ # Componentes React reutilizables  
│ │ ├── hooks/ # Custom hooks  
│ │ ├── lib/ # Utilidades y configuraciones  
│ │ └── pages/ # Componentes de página  
│  
├── server/ # Backend del sistema  
│ ├── admin/ # Panel administrativo  
│ ├── lib/ # Funciones auxiliares  
│ ├── migrations/ # Migraciones de base de datos  
│ ├── notarypro/ # Funcionalidades para notarios  
│ ├── partners/ # Servicios para socios  
│ ├── routes/ # Rutas de la API  
│ ├── seeds/ # Datos iniciales DB  
│ ├── services/ # Servicios compartidos  
│ ├── types/ # Tipos compartidos  
│ └── vecinos/ # Módulo vecinos  
│  
├── shared/ # Recursos compartidos  
│ └── schema.ts # Esquema de base de datos  
│  
├── public/ # Archivos estáticos  
└── uploads/ # Archivos subidos por usuarios

## 3. Flujos Críticos del Sistema

### Autenticación y Seguridad

* Sistema de login con tokens JWT
* Validación de rol basada en middleware
* Encriptación de passwords con bcrypt
* Cifrado SSL/TLS para todas las comunicaciones
* Sanitización de entrada para prevenir inyecciones SQL y XSS

// Archivo crítico: server/auth.ts  
export async function hashPassword(password: string) {  
 const salt = await bcrypt.genSalt(10);  
 return bcrypt.hash(password, salt);  
}  
  
export async function comparePasswords(supplied: string, stored: string) {  
 return bcrypt.compare(supplied, stored);  
}

### Verificación de Identidad

* Verificación biométrica facial (selfie)
* Escaneo NFC de cédula de identidad
* Validación de documentos oficiales
* Registro auditable de verificaciones

// Archivo crítico: client/src/lib/nfc-reader.ts  
export async function readCedulaChilena(nfcReader: NFCReader): Promise<CedulaChilenaData> {  
 const statusCallback = (status: NFCReadStatus, message?: string) => {  
 console.log(`Estado de lectura NFC: ${status}${message ? ` - ${message}` : ''}`);  
 };  
 statusCallback(NFCReadStatus.WAITING, 'Esperando tarjeta NFC...');  
   
 // Implementación de la lectura NFC...  
}

### Firma Electrónica

* Firmas electrónicas avanzadas
* Firma con certificado digital
* Verificación criptográfica
* Hash SHA-256 para documentos

// Archivo crítico: server/services/signature-service.ts  
async function generateSignatureHash(documentId, userId, timestamp) {  
 const content = `${documentId}-${userId}-${timestamp}`;  
 return crypto.createHash('sha256').update(content).digest('hex');  
}

### Video-Notarización (RON)

* Sesiones de video (Agora)
* Grabación certificada
* Control de calidad de video y audio
* Proceso dirigido por notario

// Archivo crítico: server/ron-routes.ts  
ronRouter.post('/sessions/create', async (req, res) => {  
 // Implementación de creación de sesión...  
 const sessionId = uuid();  
 const videoTokens = agoraService.generateVideoTokens(sessionId);  
 // Persistir y retornar información de sesión...  
});

### Procesamiento de Pagos

* Integración con múltiples pasarelas
* Registro de transacciones
* Notificaciones de pago
* Facturas electrónicas

// Archivo crítico: server/mercadopago-routes.ts  
mercadopagoRouter.post('/create-preference', async (req, res) => {  
 const { items, payer, backUrls } = req.body;  
 // Configuración del cliente y generación de preferencia...  
 const preference = await mercadopago.preferences.create({  
 items,  
 payer,  
 back\_urls: backUrls,  
 auto\_return: 'approved'  
 });  
   
 return res.json({ preferenceId: preference.id });  
});

## 4. Base de Datos

### Modelo de Datos Principal

┌─────────────┐ ┌──────────────┐ ┌──────────────┐  
│ users │ │ documents │ │ signatures │  
├─────────────┤ ├──────────────┤ ├──────────────┤  
│ id ├───┐ │ id │ ┌───┤ id │  
│ username │ │ │ title │ │ │ user\_id │  
│ email │ │ │ content │ │ │ document\_id │  
│ password │ │ │ status │ │ │ signature\_img│  
│ role │ └───┤ owner\_id │ │ │ timestamp │  
│ created\_at │ │ created\_at ├───┘ │ verified │  
└─────────────┘ └──────────────┘ └──────────────┘  
  
┌─────────────┐ ┌──────────────┐ ┌──────────────┐  
│ verifications│ │ ron\_sessions │ │ transactions │  
├─────────────┤ ├──────────────┤ ├──────────────┤  
│ id │ │ id │ │ id │  
│ user\_id │ │ certifier\_id │ │ user\_id │  
│ type │ │ client\_id │ │ amount │  
│ status │ │ document\_id │ │ status │  
│ meta\_data │ │ status │ │ provider │  
│ created\_at │ │ created\_at │ │ created\_at │  
└─────────────┘ └──────────────┘ └──────────────┘

### Esquema Principal (shared/schema.ts)

// Tablas principales definidas en shared/schema.ts  
  
export const users = pgTable("users", {  
 id: serial("id").primaryKey(),  
 username: text("username").notNull(),  
 password: text("password").notNull(),  
 email: text("email").notNull(),  
 fullName: text("full\_name").notNull(),  
 role: text("role").notNull().default("user"),  
 createdAt: timestamp("created\_at").defaultNow(),  
 platform: text("platform"),  
 businessName: text("business\_name"),  
 region: text("region"),  
 comuna: text("comuna"),  
 address: text("address"),  
});  
  
export const documents = pgTable("documents", {  
 id: serial("id").primaryKey(),  
 title: text("title").notNull(),  
 content: text("content"),  
 contentType: text("content\_type").default("text/html"),  
 ownerId: integer("owner\_id").references(() => users.id),  
 status: text("status").notNull().default("draft"),  
 createdAt: timestamp("created\_at").defaultNow(),  
 template: boolean("is\_template").default(false),  
});  
  
export const signatures = pgTable("signatures", {  
 id: serial("id").primaryKey(),  
 userId: integer("user\_id").references(() => users.id).notNull(),  
 documentId: integer("document\_id").references(() => documents.id).notNull(),  
 signatureImg: text("signature\_img").notNull(),  
 timestamp: timestamp("timestamp").defaultNow().notNull(),  
 verified: boolean("verified").default(false),  
 verificationMethod: text("verification\_method"),  
 signatureHash: text("signature\_hash"),  
});

## 5. Puntos críticos de seguridad

### Autenticación

* Sesiones gestionadas mediante JWT
* Verificación de roles en middleware
* Protección contra ataques de fuerza bruta
* No almacenamiento de contraseñas en texto plano

### Seguridad de Datos

* Cifrado de datos sensibles
* Validación de entrada en cliente y servidor
* Protección contra inyección SQL usando ORM Drizzle
* Backups regulares de la base de datos

### Auditoría

* Registro de todas las acciones críticas
* Logging extensivo para depuración
* Seguimiento de cambios en documentos
* Registros inmutables para cumplimiento legal

## 6. Modo QA y Funcionalidad Real

El sistema implementa un “Modo QA” especial que permite realizar pruebas end-to-end sin necesidad de interactuar con servicios externos reales:

// client/src/lib/funcionalidad-real.ts  
export function esFuncionalidadRealActiva(): boolean {  
 // Verificar si el modo funcional real está activado  
 if (typeof window !== 'undefined') {  
 return sessionStorage.getItem('modo\_funcional\_activo') === 'true' ||   
 localStorage.getItem('modo\_funcional\_activo') === 'true';  
 }  
 return false;  
}  
  
export function activarFuncionalidadReal(codigo?: string): boolean {  
 // Activar el modo funcional real  
 if (validarCodigoFuncionalReal(codigo)) {  
 sessionStorage.setItem('modo\_funcional\_activo', 'true');  
 return true;  
 }  
 return false;  
}

## 7. Integración con Dispositivos

### NFC

* Lectura de cédulas chilenas con chip NFC
* Validación de datos obtenidos
* Compatibilidad con diversos dispositivos

### Cámara Web

* Captura de imágenes para verificación
* Comparación biométrica básica
* Almacenamiento seguro de datos biométricos

### Impresión

* Generación de documentos PDF
* Códigos QR para verificación
* Impresión de certificados

## 8. Procedimientos para despliegue

### Proceso de Deploy

1. Ejecutar migraciones: npm run db:push
2. Construir aplicación: npm run build
3. Iniciar servidor: npm start

### Variables de Entorno Requeridas

DATABASE\_URL=postgresql://user:password@host:port/dbname  
JWT\_SECRET=your-jwt-secret-here  
MERCADOPAGO\_ACCESS\_TOKEN=your-mercadopago-token  
SENDGRID\_API\_KEY=your-sendgrid-key  
AGORA\_APP\_ID=your-agora-app-id  
AGORA\_APP\_CERTIFICATE=your-agora-certificate

### Mantenimiento

* Respaldos diarios de la base de datos
* Monitoreo del rendimiento del servidor
* Actualizaciones regulares de dependencias

## 9. Resolución de problemas comunes

### Problemas de NFC

* Verificar que el dispositivo tenga NFC habilitado
* Comprobar la posición correcta de la cédula
* Usar el código QA para modo de prueba si es necesario

### Errores de Video

* Verificar permisos de cámara/micrófono
* Comprobar la conexión a Internet
* Reiniciar la sesión de Agora

### Fallos en Transacciones

* Verificar estado en panel de la pasarela de pago
* Comprobar datos de tarjeta/cuenta
* Contactar al soporte de la pasarela

## 10. Contactos de Soporte

* **Soporte técnico**: soporte@vecinosexpress.cl
* **Desarrollo**: dev@vecinosexpress.cl
* **Problemas de pago**: pagos@vecinosexpress.cl

Este manual técnico proporciona una visión completa del sistema VecinosExpress. Para funcionalidades específicas, consulte la documentación detallada de cada módulo en la carpeta docs/.