DOCUMENTO FUNCIONAL DEL PROYECTO

---

1. Introducción y contexto

Objetivo: desarrollar una aplicación web y móvil para la **gestión integral de comunidades de vecinos** que centralice comunicación, economía, mantenimiento, organización de juntas, seguridad, documentación y contactos útiles.

Problema/Necesidad: las comunidades suelen tener **mala comunicación**, **falta de transparencia** en gastos, **desorganización** de incidencias y **pérdida de información** (actas, contratos, presupuestos) en canales dispersos (papel, WhatsApp, emails).

Usuario/Cliente final:

- **Vecinos/propietarios**: consultar cuotas, votar, ver anuncios, reportar incidencias.
- **Administradores**: gestionar economía, contratos, juntas y documentación.
- **Cargos de la comunidad** (presidente/secretario): convocatorias, actas, votaciones.
- **Proveedores**: mantenimiento y seguimiento de incidencias asignadas.

Solución propuesta y propósito: VeciApp (nombre provisional) centraliza todo en un único sistema: tablón, chat, encuestas, economía (cuotas, presupuestos, balances), mantenimiento (incidencias, agenda, contratos), organización de juntas (convocatorias, votaciones online, actas), seguridad (avisos), repositorio documental e **índice de contactos** (mantenimiento, seguros, emergencias, proveedores, ayuntamiento). Mejora la **transparencia**, **participación** y **eficiencia**.

---

2. Análisis de requisitos

2.1. Requisitos funcionales (RF)

Código	Descripción del requisito funcional
RF1	Registro e inicio de sesión de usuarios (vecinos, administradores, proveedores).
RF2	Tablón de anuncios digital: crear, listar, comentar y fijar anuncios por comunidad.
RF3	Chat entre vecinos y/o por grupos (por portal/escala/tema) con adjuntos.
RF4	Encuestas rápidas con opciones múltiples y resultados en tiempo real.
RF5	Consulta de cuotas individuales y estado de pagos.
RF6	Acceso a presupuestos, gastos y balances de la comunidad.
RF7	Registro de incidencias (descripción/fotos), asignación y seguimiento de estado.
RF8	Agenda de mantenimiento (preventivo/correctivo) con recordatorios automáticos.
RF9	Gestión de contratos (alta, vencimientos, renovaciones, adjuntos).
RF10	Convocatorias de juntas con envío de avisos y documentos adjuntos.
RF11	Votaciones online con control de quórum y trazabilidad de votos.
RF12	Actas digitales: generación/subida, firma electrónica y archivo histórico.
RF13	Avisos de seguridad (robos/situaciones sospechosas) con alertas push o email.
RF14	Repositorio documental con carpetas por año/tema y control de versiones.
RF15	Histórico de juntas y actas accesible y filtrable.
RF16	Directorio de contactos: mantenimiento, seguros, emergencias, proveedores, ayuntamiento.
RF17	Sistema de notificaciones (in-app, push, email) configurable por usuario.

RF18	Roles y permisos (vecino, presidente, administrador, conserje, proveedor).

2.2. Requisitos no funcionales (RNF)

Cómo debe comportarse el sistema.
Incluye aspectos como rendimiento, seguridad, compatibilidad o facilidad de uso.

Código	**Descripción del requisito no funcional**
RNF1	Uso fluido para dispositivos móviles
RNF2	Las páginas deberán cargarse en menos de tres segundos.
RNF3	Escalabilidad para múltiples comunidades
RNF4	Disponibilidad objetivo ≥ 99,5%
RNF5	Protección de datos(GDPR /LOPDGDD):consentimientos,derechos ARCO,retención.
RNF6	Seguridad reforzada para documentos sensibles como ,actas, facturas y votaciones
RNF7	Trazabilidad de notificaciones y entregas (emails,push,SMS)

2.3. Restricciones

Lenguajes/tecnologías:

- Frontend: **React Native** (móvil).
- Backend: **Node.js + Express** (TypeScript) con **PostgreSQL** y **Prisma**.

Recursos: equipo pequeño, tiempo limitado (curso).

Dependencias técnicas: alojamiento cloud básico (PaaS), almacenamiento de ficheros tipo S3.

3. Análisis de usuarios y roles

Objetivo: identificar quién usará el sistema y qué podrá hacer.

Describe los distintos tipos de usuario, sus necesidades y sus permisos.

Rol	**Descripción**	**Permisos principales**
Administrador	Gestor profesional de la app con permisos completos.	Alta/baja usuarios, comunidades, economía, contratos, convocatorias, actas, configuración.
Presidente/ Secretario	Cargos de la comunidad.	Convocar juntas, validar actas, lanzar votaciones, publicar anuncios.
Vecino/ Propietario	Usuario residente o propietario.	Ver anuncios, votar, registrar incidencias, consultar cuotas/documentos, chatear.
Conserje	Operaciones diarias del edificio.	Ver/actualizar incidencias y agenda, publicar avisos de servicio.

4. Casos de uso / Escenarios de uso

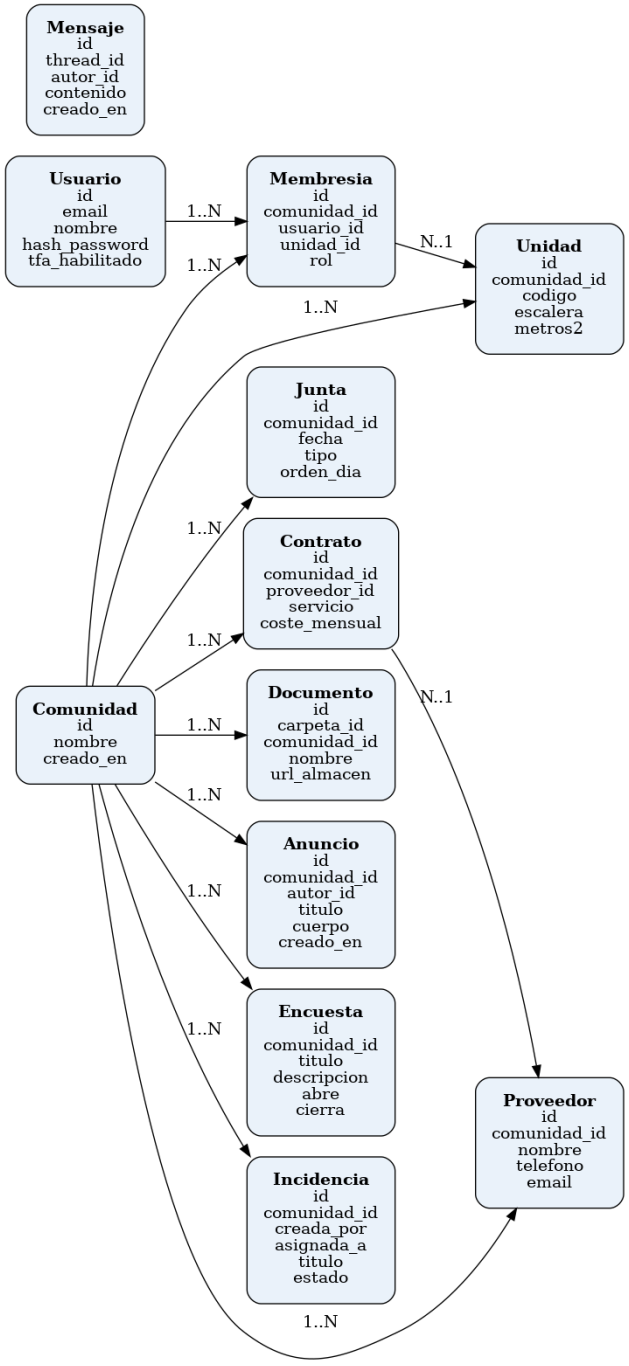
Objetivo: mostrar cómo interactúan los usuarios con el sistema.

Selecciona de tres a cinco casos principales y descríbelos brevemente.

Código	**Nombre del caso de uso**	**Actor principal**	**Descripción**	**Resultado esperado**
CU1	Publicar anuncio	Administrador	Publica aviso en el tablón con adjuntos.	Anuncio visible y notificado a la comunidad.
CU2	Registrar incidencia	Vecino	Reporta incidencia con fotos y prioridad.	Incidencia creada, asignada y notificada.
CU3	Votar en junta	Vecino	Recibe convocatoria, accede y emite voto.	Voto registrado.
CU4	Consultar cuotas	Vecino	Revisa estado de pagos y próximas cuotas.	Información económica visible y exportable.
CU5	Subir acta	Secretario	Sube/genera el PDF del acta firmada.	Acta archivada y versionada en el repositorio.

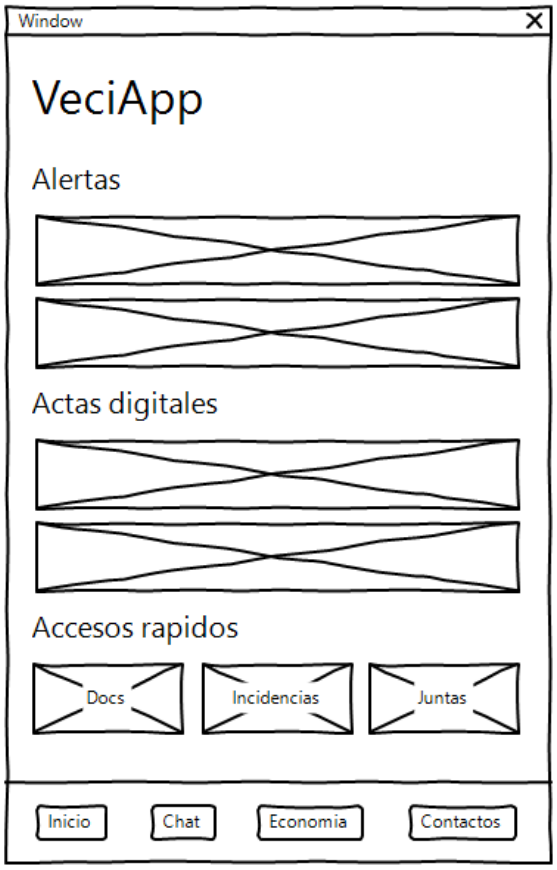
---

5. Modelo de datos o estructura de la información

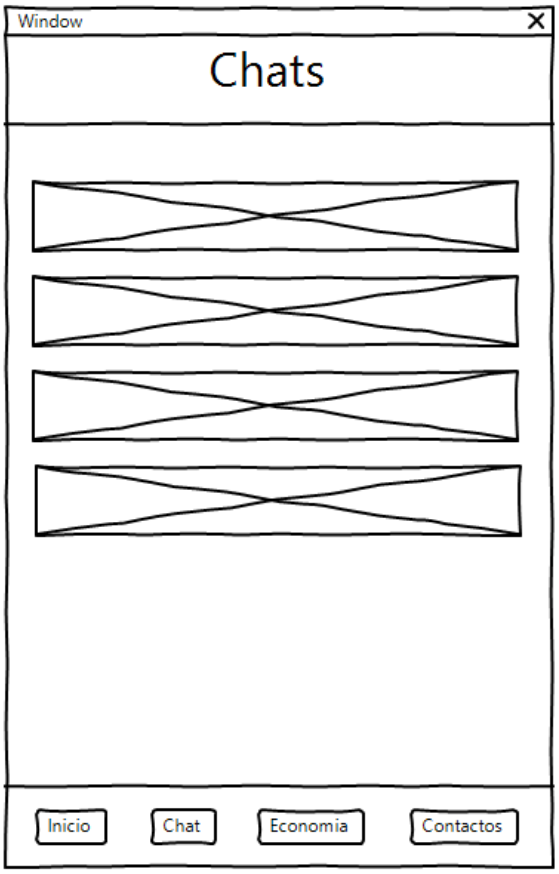


6. Diseño de la interfaz

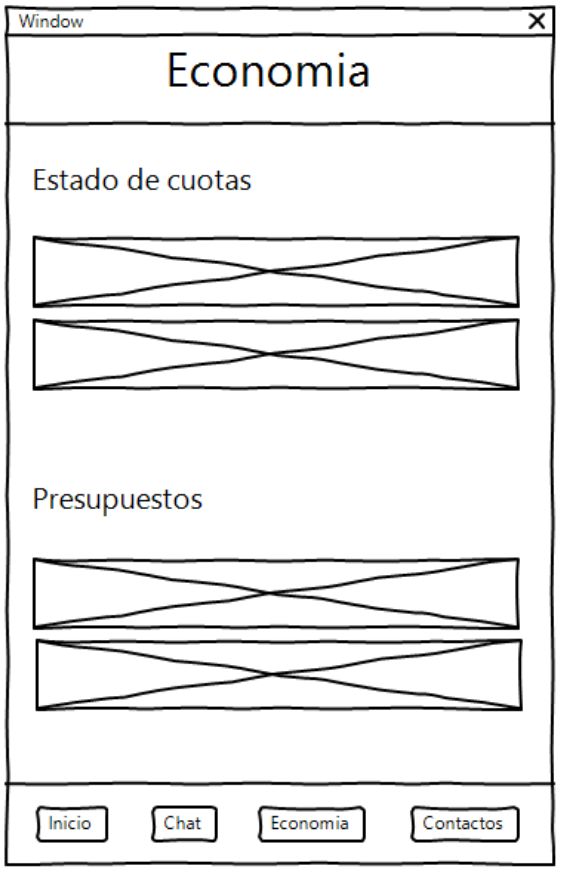
- Inicio/Tablón: feed de anuncios y alertas, accesos rápidos a incidencias, juntas y documentos.



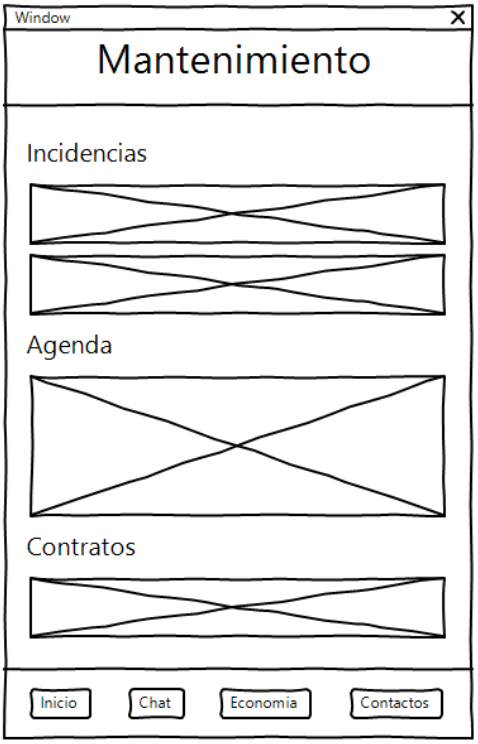
- Chat/Grupos: hilos por tema/portal; envío de fotos/documentos.



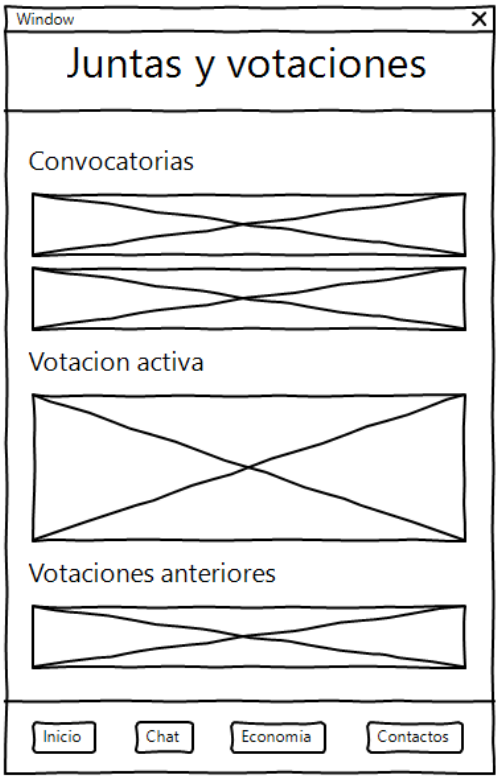
- Economía: cuotas personales, presupuestos, gastos y balances (filtros y exportación).



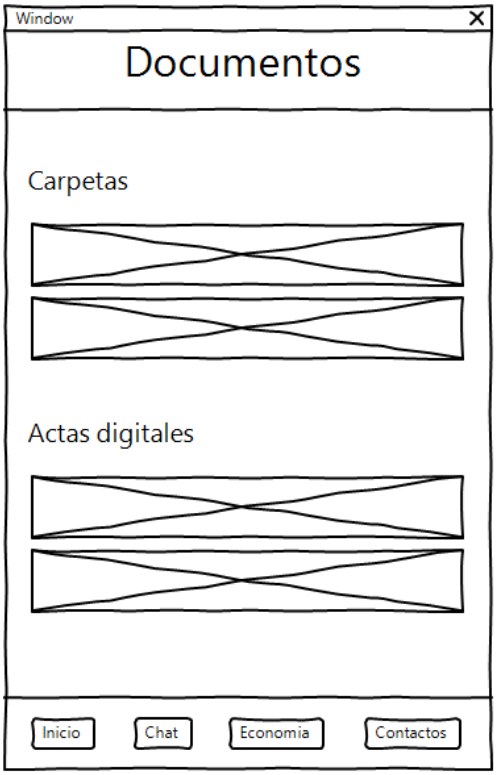
- Mantenimiento: listado y detalle de incidencias, agenda y contratos con vencimientos.



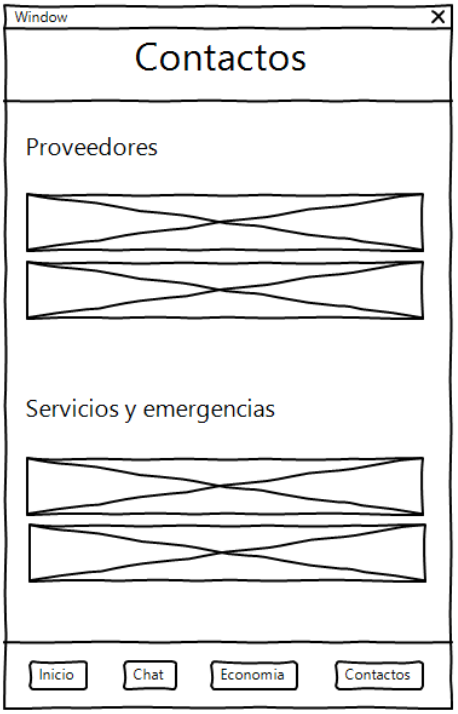
- Juntas y votaciones: convocatorias, emisión de voto y resultados.



- Documentos: repositorio por carpetas, histórico de actas.



- Contactos: proveedores, seguros, emergencias, ayuntamiento.



---

7. Planificación técnica

Lenguajes y frameworks:

- Framework principal: React Native (TypeScript)
    - Permite desarrollar una sola base de código para Android e iOS.
    - Ofrece buen rendimiento, integración con APIs nativas y una gran comunidad.
- Gestión de navegación: React Navigation (para moverse entre pantallas).
- Gestión de estado: Zustand o Redux Toolkit (para manejar datos globales como usuario, comunidad, incidencias, notificaciones, etc.).
- Validación y formularios: React Hook Form + Zod, asegurando validaciones tipadas.
- Almacenamiento local: AsyncStorage o SQLite (para datos temporales y uso offline).
- Notificaciones push: Expo Notifications o OneSignal, para alertas inmediatas (incidencias, votaciones, seguridad)-
- Autenticación: JWT (tokens) gestionados por el backend, con soporte de OAuth2 si se necesita login externo.
- Diseño UI: Tailwind React Native Classnames (estilo rápido, coherente y responsive).

Backend(API y servicios)

- Backend (API y servicios)
- Runtime: Node.js + Express (TypeScript).
- Base de datos: PostgreSQL (gestionada en la nube).
- ORM: Prisma (para consultas seguras y migraciones).
- Comunicación en tiempo real: Socket.IO, para chat, notificaciones y actualizaciones instantáneas.
- Almacenamiento de archivos: S3-compatible (para documentos, fotos de incidencias, actas, etc.).
- Colas de procesos: BullMQ (Redis), para tareas en segundo plano (envío de correos, generación de PDFs, recordatorios).
- Autenticación y roles: JWT + middleware de roles (vecino, presidente, administrador, proveedor).

Infraestructura y herramientas

- Control de versiones: Git + GitHub.
- Integración continua (CI/CD): GitHub Actions (test, build y despliegue automático).
- Despliegue backend: Render, Fly.io o AWS (servicio PaaS).
- Base de datos gestionada: Supabase o RDS (PostgreSQL).
- Almacenamiento de archivos: Cloudflare R2 o AWS S3.
- Monitoreo y errores: Sentry (seguimiento de fallos y logs en producción).
- Testing: Jest (unitario), Detox (E2E móvil), Supertest (API).

Reparto de tareas(Provisional)

Responsable	Área principal
Santi	Backend (API REST, autenticación, incidencias, votaciones).
Edgar/Santi	App móvil (interfaz, chat, notificaciones, mantenimiento).
Edgar	Integración con base de datos, pruebas y despliegue.

---

8. Análisis de riesgos

8.1. Identificación de riesgos

- Falta de tiempo / solapamiento con otras asignaturas.
- Complejidad de votaciones (quórum, auditoría).
- Protección de datos (GDPR) y gestión de documentos sensibles.
- Integración de notificaciones push/SMS.

- Pérdida de datos o errores en migraciones.

8.2. Valoración y respuesta

Clasifica cada riesgo según su probabilidad e impacto, e indica cómo se mitigará.

Riesgo	**Probabilidad**	**Impacto**	**Plan de prevención o contingencia**
Falta de tiempo	Alta	Alta	Sprints cortos, entregas intermedias, priorizar MVP.
Votaciones complejas	Media	Alta	Definir reglas y casos limites antes de programar;tests E2E
GDPR/privacidad	Media	Alta	Minimización de datos,roles estrictos, cifrado y auditoría.
Notificaciones	Media	Media	Implementar primer push mobil(Android/IOS) amb OneSignal o EXPO;
Pérdida de datos	Baja	Alta	Backups diarios de la base de datos,versionado en S3 y pruebas de restauración.

---

9. Validación y criterios de éxito

Criterios de aceptación:

- Usuarios pueden **iniciar sesión**, ver **tablón**, **crear y seguir incidencias**, **votar** y **descargar actas**.
- Los administradores gestionan **cuotas, gastos y presupuestos** y publican **convocatorias**.
- **Notificaciones** llegan según preferencia del usuario.

Pruebas previstas:

- Unitarias (servicios), integración (API con supertest), E2E (Playwright).
- Pruebas de usabilidad con 3–5 usuarios (tareas: reportar incidencia, votar, consultar cuota).
- Pruebas de rendimiento (listas de mensajes/documentos) y de seguridad básica.

Indicadores de calidad:

- Tasa de participación en votaciones ≥ 60% (objetivo).
- Tiempo medio de resolución de incidencias.
- Reducción de consultas repetidas al administrador (medida por uso del repositorio).

---

10. Conclusión

Decisiones clave:
El proyecto se desarrollará exclusivamente como una aplicación móvil, utilizando React Native (con TypeScript) para crear una app multiplataforma para Android e iOS.
El backend estará implementado con Node.js + Express, y la información se gestionará con una base de datos PostgreSQL.

La aplicación incluirá los módulos principales: comunicación (tablón, chat, encuestas), gestión económica (cuotas, presupuestos, gastos), mantenimiento (incidencias, agenda y contratos), organización de juntas (convocatorias, votaciones y actas digitales), seguridad (alertas) y documentación (repositorio e histórico). Todo el sistema estará diseñado con multitenancy (una única plataforma capaz de gestionar múltiples comunidades) y con roles y permisos (RBAC), además de un sistema de auditoría para registrar acciones importantes.

Valor aportado:
La app ofrece una gestión integral accesible desde el móvil, mejorando la comunicación y la transparencia entre vecinos y administradores. Permite reducir tiempos de respuesta, centralizar toda la información y facilitar procesos como votaciones, incidencias, mantenimientos o consulta de documentos, todo desde una única aplicación que los usuarios pueden llevar siempre consigo.

Próximos pasos:

1. Configurar el entorno de desarrollo móvil (React Native, dependencias y navegación).
2. Crear el repositorio e implementar la arquitectura inicial del backend y la base de datos.
3. Desarrollar los casos de uso prioritarios:
    - tablón de anuncios,
    - chat e incidencias,
    - votaciones y convocatorias,
    - repositorio documental.

4. Conectar la app con el backend e implementar notificaciones push.

5. Realizar pruebas internas y preparar una versión beta móvil para Android/iOS.