MANUAL DE USUARIO Y REFERENCIA TÉCNICA

Sistema Red Social para comunidad de lectores.

Versión: 2.0

Fecha: 17-11-2025

Audiencias: Usuarios finales, administradores funcionales y soporte técnico.

Estado del branch revisado: Development.

> Este manual consolida la operación funcional y aspectos técnicos clave del sistema (plataforma social de comunidades, libros, chat y notificaciones) construido con React + Firebase + TailwindCSS + Bootstrap. Incluye instalación, configuración, arquitectura, flujos, seguridad, solución de problemas, despliegue y anexos.

ÍNDICE

1. Introducción y Objetivos

El sistema permite gestionar comunidades temáticas, catálogo de libros, interacción social mediante chat público y privado, notificaciones de amistad y perfiles de usuario. Objetivo: brindar una experiencia colaborativa centralizada y extensible.

2. Alcance y Público Objetivo

- Usuarios generales: crear cuenta, participar en comunidades, añadir libros, conversar.

- Moderadores/Administradores (futuro): gestionar comunidades y reportes.

- Soporte técnico: mantener despliegues, aseguramiento de seguridad.

3. Visión General Funcional

El usuario se autentica, navega por la Navbar, consume feeds de comunidades, añade libros a su biblioteca, chatea en tiempo real y gestiona amistades y notificaciones.

4. Tecnologías y Dependencias Clave

Extracto de package.json:

- React 19 (react, react-dom), react-scripts 5 (Create React App).
- Enrutamiento: react-router-dom v7.
- Formularios: react-hook-form.
- UI: bootstrap, react-bootstrap, tailwindcss (branch Tailwind).
- Firebase SDK ^11.9.0 (Auth, Firestore/Storage).
- Observabilidad: @sentry/react, @sentry/tracing, web-vitals.
- Testing: @testing-library/*.
- Utilidades: browser-image-compression, react-icons, react-modal.

5. Requisitos Previos

- Node.js LTS >=16 (recomendado >=18). Verificar con node -v.
- npm >=8 (npm -v).
- Cuenta y proyecto configurado en Firebase (Auth, Firestore, Storage activados).
- Acceso a clave de Sentry (opcional) para monitoreo.

6. Preparación del Entorno

PowerShell (Windows):

cd C:\Users\diego\Documents\WEB\Integrador\avance3\Integrador

##El path depende de donde se haya guardado el archivo.

npm install

Si existe conflicto: eliminar node_modules y package-lock.json y reinstalar.

7. Instalación y Scripts NPM

Scripts principales:

```
npm start     # Dev server (hot reload)
npm run build  # Build producción (carpeta build/)
npm test      # Suite de pruebas (si se agregan tests)
npm run eject  # Extrae configuración CRA (irreversible)
```

8. Variables de Entorno y Configuración Firebase

Crear un archivo .env (no versionado) con claves:

```
REACT_APP_FIREBASE_API_KEY=...
REACT_APP_FIREBASE_AUTH_DOMAIN=...
REACT_APP_FIREBASE_PROJECT_ID=...
REACT_APP_FIREBASE_STORAGE_BUCKET=...
REACT_APP_FIREBASE_MESSAGING_SENDER_ID=...
REACT_APP_FIREBASE_APP_ID=...
```

En src/Firebase/config.js importar estas variables usando process.env. Mantener fuera del control público.

8.1 Guía Paso a Paso de Configuración Firebase

1. Crear proyecto en consola Firebase.
2. Activar Authentication (Email/Password).
3. Crear base de datos Firestore (modo de pruebas inicial o producción).
4. Definir reglas iniciales (endurecer más tarde):

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
```

```
   match /users/{userId}{
    allow read: if true;
    allow write: if request.auth != null && request.auth.uid == userId;
   }
   match /communities/{cid}{
    allow read: if true;
    allow write: if request.auth != null;
   }
   match /posts/{postId}{
    allow read: if true;
    allow create: if request.auth != null;
    allow update, delete: if request.auth != null && request.resource.data.authorUid ==
request.auth.uid;
   }
   match /messages/{mid}{
    allow read, write: if request.auth != null &&
     ((request.auth.uid == resource.data.fromUid) || (request.auth.uid ==
resource.data.toUid));
   }
   match /friendRequests/{frid}{
    allow read, write: if request.auth != null &&
     (request.auth.uid == resource.data.fromUid || request.auth.uid ==
resource.data.toUid);
   }
  }
}
```

5. Activar Storage y reglas básicas:

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
```

```
   match /images/{allPaths=**} {
     allow read: if true;
     allow write: if request.auth != null;
   }
 }
}
```

6. Registrar aplicación Web y copiar credenciales a .env.

7. Implementar config.js:

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';
import { getStorage } from 'firebase/storage';

const firebaseConfig = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_APP_ID,
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const db = getFirestore(app);
export const storage = getStorage(app);
```

8. Smoke test: iniciar sesión y hacer console.log(auth.currentUser); leer documento en users.

9. Endurecer reglas para producción (quitar lecturas públicas sensibles, validar tamaños en Storage).

10. Añadir monitoreo (Sentry) para errores Firebase.

Checklist:

- [ ] Proyecto creado

- [ ] Auth activo

- [ ] Firestore con reglas

- [ ] Storage activo

- [ ] .env completo

- [ ] config.js implementado

- [ ] Smoke test OK

9. Arquitectura (Capas y Principios)

- Presentación: Componentes React (funcionales) + Hooks.

- Estado global: Contexto AuthContext para sesión; resto distribuido.

- Enrutamiento: Definido en App.jsx con componentes de rutas privadas/públicas.

- Estilos: Tailwind + CSS modular heredado (src/CSS/). Migración progresiva al paradigma utilitario.

- Observabilidad: reporteWebVitals.js + inicialización Sentry (Observability/).

Principios:

- Separación de responsabilidades por carpeta de dominio.

- Reutilización de componentes de formulario y feed.

- Acceso autorizado a rutas críticas (perfil, chat, comunidad privada).

10. Estructura de Carpetas Detallada (Resumen)

src/

App.jsx # Layout y rutas

index.js # Punto de entrada

index.css # Estilos globales

Componentes/

Comunidades/ ... # Gestión de comunidades y feed

Libro/ ... # Catálogo y detalle de libros

Chat/ ... # Mensajería

Perfil/ ... # Perfil propio y público

Sesion/ ... # Auth (Registro/Login/Context)

Navbar/ ... # Barra de navegación

Notificaciones/ ... # Solicitudes y avisos

RutaPrivada/ ... # Wrapper de autenticación

RutaPublica/ ... # Rutas accesibles sin login

CSS/ # Hojas de estilo tradicionales

Firebase/config.js # Config Firebase

Observability/ # Sentry y Web Vitals

helpers/imports.js # Re-exports utilitarios

11. Enrutamiento y Control de Acceso

RutaPrivada.jsx verifica estado de user desde AuthContext; si no autenticado, redirige a login. RutaPublica.jsx puede impedir acceso a páginas de login si ya autenticado.

12. Autenticación y Sesión (AuthContext)

AuthContext.jsx expone:

- user: objeto usuario actual.

- Métodos: login(email, pass), logout(), register(data) (depende implementación), onAuthStateChanged (interno).

Mantener mínima información sensible en memoria; para datos extra usar Firestore.

13. Roles y Permisos

Estado actual: Rol implícito único (usuario).

Extensión futura:

- admin: crear/eliminar comunidades, moderar publicaciones.

- moderator: editar reglas y gestionar miembros.

Implementación sugerida: campo role en colección users y lógica condicional en componentes.

14. Módulos Funcionales

Navbar

Navegación entre dominios. Incluye cierre de sesión. Debe reflejar estado (mostrar opciones adicionales autenticado).

Inicio

Resumen general (puede incluir feed global o recomendaciones). Entrada rápida a acciones comunes.

Comunidades

Operaciones:

- Crear (formulario: nombre, descripción, imagen, reglas).
- Editar (EditComunidad.jsx).
- Listar (ListaComunidades.jsx).
- Ver descripción (Descripcion.jsx).
- Feed (ComunidadFeed.jsx) con publicaciones (Publicacion.jsx).
- Crear publicación (CrearPublicacion.jsx).
- Miembros (MiembrosComunidad.jsx).

Libros

- Añadir (AddLibro.jsx): título, autor, género, sinopsis, portada.
- Editar (EditLibro.jsx).
- Listar (ListaLibros.jsx).
- Detalle (LibroDetail.jsx / LibroDetailContainer.jsx).

Perfil

- Propio (Perfil.jsx): editar bio, avatar, ver bibliotecas/comunidades.
- Público (PerfilPublico.jsx): vista restringida sin edición.

Chat

- Público (chat.jsx): canal general.

- Privado (PrivateChatPage.jsx): conversación directa.

Mensajes: texto plano (extensible a imágenes).

Considerar añadir timestamps y estado (entregado/leído) futuro.

Notificaciones

NotificacionesAmistad.jsx y ModalSolicitudes.jsx gestionan solicitudes de amistad y acciones aceptar/denegar.

Rutas Privadas/Públicas

Controlan acceso para reforzar seguridad y UX consistente.

15. Flujos Paso a Paso (Detalle)

Registro

1. Abrir página Registro.

2. Completar correo y contraseña.

3. Enviar; esperar confirmación (si se requiere email verification).

4. Iniciar sesión y completar perfil.

Crear Comunidad

1. Navbar → Comunidades.

2. Botón Crear Comunidad.

3. Formulario + Validar campos.

4. Guardar; redirigir al feed específico.

5. Crear primera publicación opcional.

Añadir Libro

1. Ir a sección Libros / Añadir.

2. Completar metadatos.

3. Comprimir imagen (browser-image-compression se encarga).

4. Guardar y validar aparición en lista.

Mensaje Privado

1. Abrir Chat Privado.

2. Seleccionar destinatario.

3. Escribir y enviar.

4. Confirmar persistencia y recepción.

16. Modelo de Datos (Sugerido Firebase)

Colecciones:

- users: { uid, displayName, email, photoURL, bio, role }

- communities: { id, name, description, rules, ownerUid, createdAt }

- communityMembers: { communityId, userUid, joinedAt, role }

- posts: { id, communityId, authorUid, content, createdAt }

- books: { id, title, author, genre, synopsis, coverUrl, ownerUid }

- messages: { id, conversationId, fromUid, toUid (privado), text, createdAt }

- friendRequests: { id, fromUid, toUid, status, createdAt }

Indexar por campos consultados (communityId, ownerUid, toUid) para eficiencia.

17. Validaciones y Mensajes de Error

Ejemplos:

- Registro: contraseña < 6 caracteres → mensaje "Contraseña demasiado corta".

- Crear comunidad: nombre vacío → "El nombre es obligatorio".

- Añadir libro: título duplicado (si se controla) → "Ya existe un libro con ese título".

- Chat: mensaje vacío → botón deshabilitado.

18. Seguridad y Privacidad

- No exponer claves Firebase en repositorios públicos.

- Reglas Firestore: restringir escritura a usuarios autenticados; permitir lectura pública selectiva (p.ej. libros).

- Sanitizar contenido de publicaciones y mensajes (evitar XSS).

- Limitar tamaño de imágenes y validar tipos MIME.

- Configurar Sentry sin enviar datos PII.

19. Rendimiento y Observabilidad

- web-vitals para métricas (LCP, FID, CLS).

- Sentry: captura errores y performance traces.

- Optimización imágenes: browser-image-compression.

- Lazy loading potencial para rutas pesadas (pendiente de implementación con React.lazy).

20. Estilos y Personalización

- Tailwind: utilidades rápidas (branch Tailwind).

- tailwind.config.js: extender paleta y tipografías.

- CSS heredado en src/CSS/ para componentes clásicos; migrar gradualmente.

21. Accesibilidad y Usabilidad

Recomendaciones:

- Etiquetas aria-* en formularios y botones clave.

- Contraste adecuado (ver WCAG AA).

- Focus visible para navegación por teclado.

- Texto alternativo en imágenes de portada y avatar.

22. Despliegue

Netlify/Vercel

1. Ejecutar npm run build.

2. Subir carpeta build/ como deploy estático.

3. Configurar variables de entorno en panel.

Firebase Hosting

1. npm install -g firebase-tools.

2. firebase init hosting (seleccionar carpeta build).

3. firebase deploy.

23. Mantenimiento y Versionado

- Uso de branches descriptivos: feature/comunidades, fix/chat-duplicados, style/tailwind-migracion.

- Revisar dependencias cada trimestre (npm outdated).

- Actualizar manual con cambio de versión mayor.

24. Solución de Problemas y FAQ

Problema | Acción rápida

npm install falla | Eliminar node_modules y package-lock.json; reinstalar.

Error Firebase Auth | Verificar claves y habilitar método de autenticación.

Imágenes no suben | Validar tamaño <2MB y permisos Storage.

Chat no actualiza | Revisar conexión y listeners (suscripciones).

Build lento | Analizar peso de imágenes y activar gzip en hosting.

FAQ:

- *¿Cómo agrego un rol admin? Añadir campo role y condicionar UI.*

- *¿Puedo recuperar contraseña? Implementar sendPasswordResetEmail de Firebase (no incluido aún).*

- *¿Puedo extender mensajes a imágenes? Añadir campo imageUrl en messages y subida a Storage.*

25. Buenas Prácticas Operativas

- Revisar logs Sentry semanalmente.

- Purga de datos huérfanos (comunidades sin owner) mensual.

- Documentar nuevas rutas antes de fusionar PR.

26. Glosario

- Feed: Lista cronológica de publicaciones de una comunidad.

- Contexto: Mecanismo React para compartir estado global.

- Role: Nivel de permiso asignado al usuario.

- Build: Versión optimizada lista para producción.

- Web Vitals: Métricas esenciales de experiencia usuario.

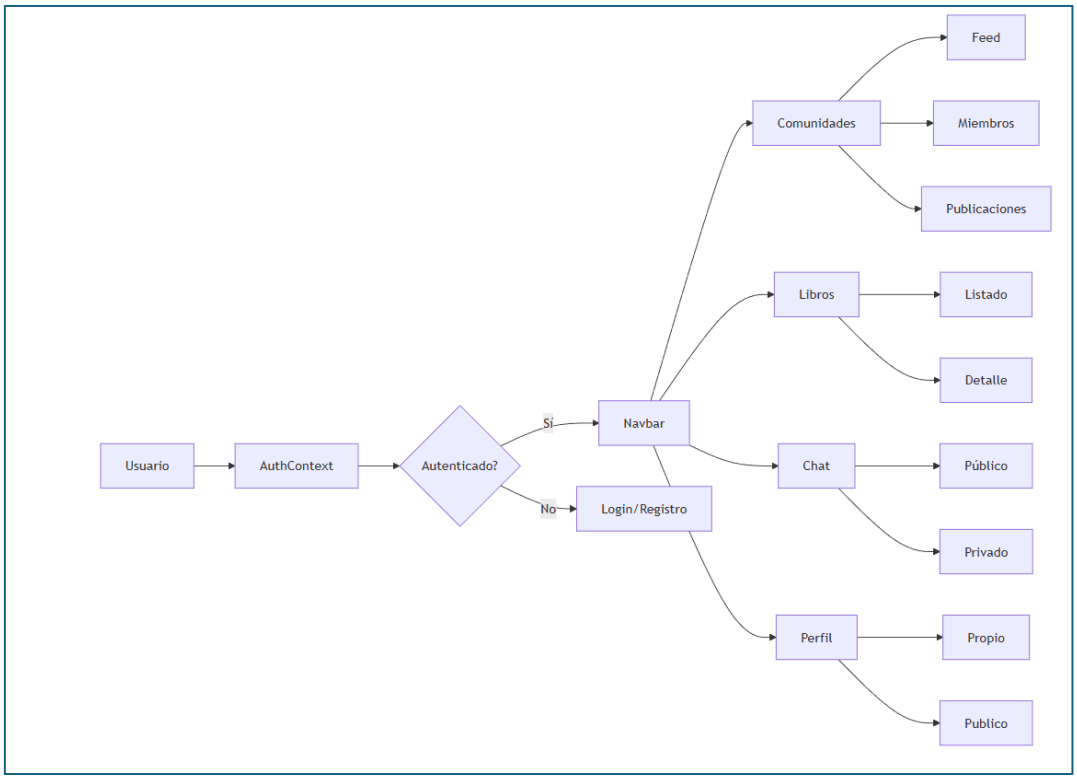
27. Anexos Técnicos

Diagrama Lógico (Texto)

Usuario → Autenticación (Firebase Auth) → Acceso a: Comunidades / Libros / Chat / Notificaciones.

Comunidades ↔ Posts ↔ Miembros.

Chat (conversaciones privadas) ↔ Mensajes.



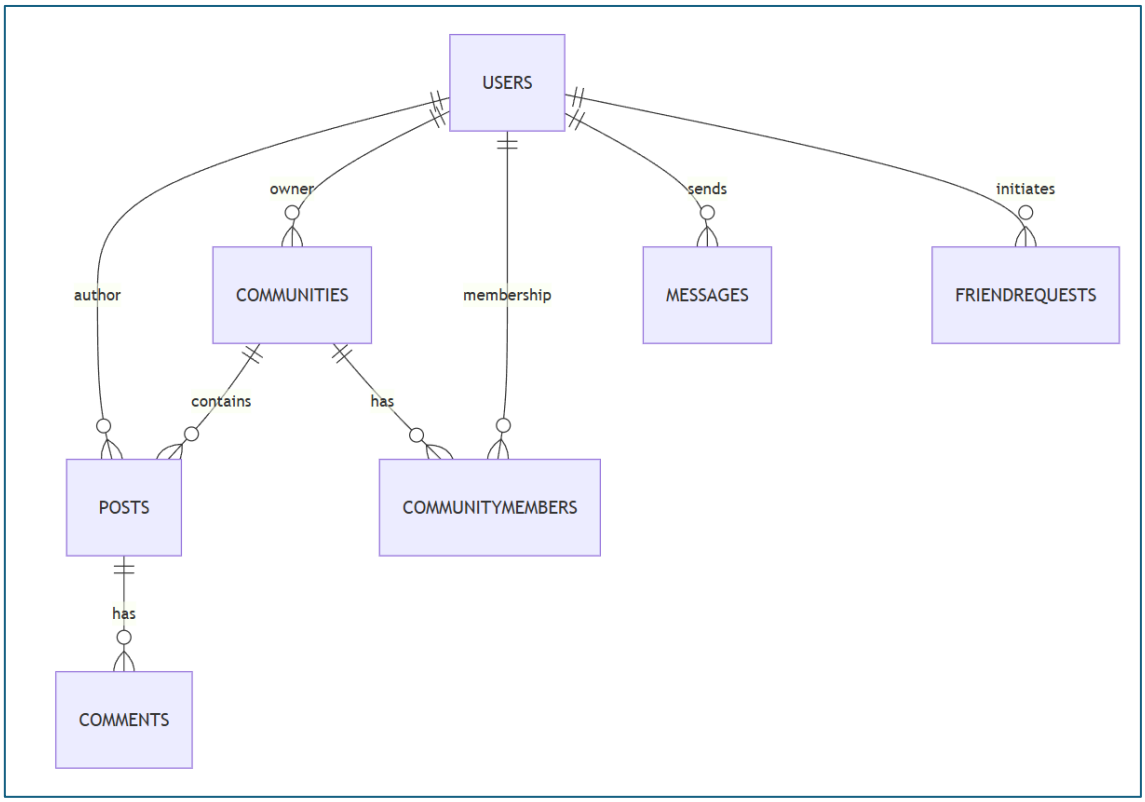


Diagrama PlantUML