MINI HACKATON "APLICACION PARA HOJAS DE VIDA L.A.M.A"

PROGRAMACIÓN DISTRIBUIDA 190304014-1

DANIEL ANDREY VILLAMIZAR ARAQUE

INTEGRANTES:

LAURA ESCOBAR ROJO ZIUVAR RUIZ ÁLVAREZ

INSTITUTO TECNOLOGICO METROPOLITANO 2025-II

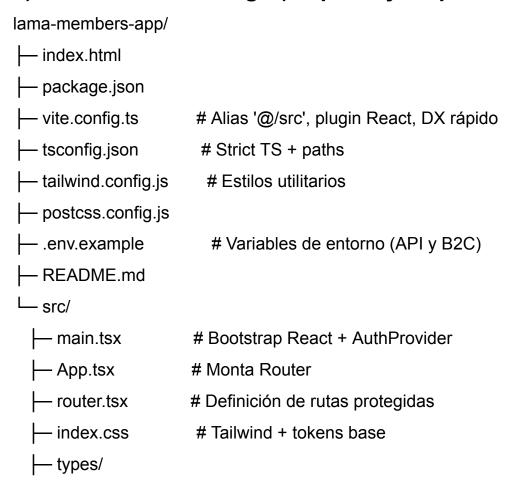
1) Propósito y alcance

Problema a resolver: gestionar miembros (alta, edición, baja, búsqueda y análisis rápido) con autenticación social (vía Azure AD B2C) y capacidad de trabajar en clase sin backend gracias a modo demo.

Qué ofrece hoy: UI React + TS lista para producción, autenticación MSAL, CRUD contra API REST (o localStorage en demo), filtros combinados, importación/exportación CSV, dashboard de métricas y estructura limpia para escalar.

Diseño para el mini-hackatón: separa responsabilidades por capas, facilita que cada sub-equipo tome un módulo (auth, UI, servicios, dashboard, import/export) y muestren una demo funcional en minutos.

2) Estructura del código (carpetas y responsabilidades)



```
└─ miembro.ts
                 # Tipado fuerte del dominio
- auth/
— msal.ts
               # Config y creación de MSAL PublicClientApplication
AuthProvider.tsx # Contexto MSAL para la app
PrivateRoute.tsx # Guardia de rutas: loginRedirect + Navigate
- services/
— api.ts
          # Axios + inyección de Bearer (B2C)
— members.ts
                 # Endpoints de Miembros + fallback demo/seed
utils/
☐ date.ts # Helpers de fecha (ISO)
- components/
─ PageShell.tsx
                 # Layout base (navbar + container)
─ SearchBar.tsx
                  # Filtros reactivos (q/rango/estatus/ciudad)
# Tabla CRUD (acciones Editar/Borrar)
─ MemberForm.tsx
                   # Form con RHF + Zod (validaciones)
☐ ImportExport.tsx # Botones y lógica CSV
pages/
MembersList.tsx # Lista + filtros + import/export + borrar
 MemberEdit.tsx
                  # Crear/Editar (navega por id)
 Dashboard.tsx
                 # KPIs: totales, activos/inactivos, por rango
               # Claims del usuario autenticado
Profile.tsx
└─ NotFound.tsx
```

Principios aplicados

Separación de capas: auth, services (datos), components (UI reusable), pages (escenas), types (dominio).

Tipado fuerte: types/miembro.ts define el contrato de datos de extremo a extremo.

Dependencias aisladas: services/api.ts centraliza Axios, headers y token MSAL; el resto del código no "sabe" de tokens.

Rutas protegidas: PrivateRoute fuerza loginRedirect y evita pantallas sin credenciales.

UX robusta en clase: modo demo en services/members.ts con seed de 3 registros + localStorage para CRUD.

3) Datos y dominio

Tipo Miembro (resumen de campos clave):

Identificación y contacto: id, nombre, apellido, correoElectronico, celular, ciudad, direccion, contactoEmergencia.

Estado y categoría: rango (Full Color|Rockets|Prospect), estatus (Activo|Inactivo|Suspendido).

Moto/detalles: moto, marca, cilindrajeCC, anoModelo, placaMatricula.

Otros (opcionales): fechalngreso, fechaNacimiento, cedula, rh, eps, padrino, foto, etc.

Validaciones de formulario (Zod)

nombre/apellido: requeridos.

correoElectronico: formato email (opcional).

cilindrajeCC: numérico entero positivo (opcional).

Extensible: agrega reglas sin tocar la UI gracias a zodResolver.

4) Autenticación (Azure AD B2C + MSAL)

Configuración: auth/msal.ts recibe VITE_B2C_CLIENT_ID, VITE_B2C_AUTHORITY, VITE_B2C_KNOWN_AUTHORITY desde .env.local.

Integración: el AuthProvider rodea toda la app y expone contexto MSAL.

Protección de rutas: PrivateRoute verifica useIsAuthenticated(). Si no hay sesión:

dispara instance.loginRedirect()

retorna <Navigate/> para evitar flickers y contenidos protegidos a medio render.

Tokens en API: en services/api.ts el interceptor intenta un acquireTokenSilent() y, si lo obtiene, inyecta Authorization: Bearer <accessToken>.

Nota: la app está lista para User Flows SignUpSignIn en B2C (puedes activar social login como Google/Facebook en el portal de Azure). En clase, si no configuras B2C, la UI seguirá levantando, pero las rutas protegidas intentarán redirigir a login.

5) Acceso a datos (servicios y modo demo)

Servicios REST (services/members.ts):

listMembers(params?) → GET /miembros con q, rango, estatus, ciudad.

getMember(id) → GET /miembros/:id

createMember(body) → POST /miembros

updateMember(id, body) → PUT /miembros/:id

deleteMember(id) → DELETE /miembros/:id

Fallback demo

Si el GET falla, se llama demoSeedlfNeeded() y se usa localStorage (lama-members) con 3 registros semilla.

En MembersList.tsx y MemberEdit.tsx se capturan errores para continuar operando sin backend:

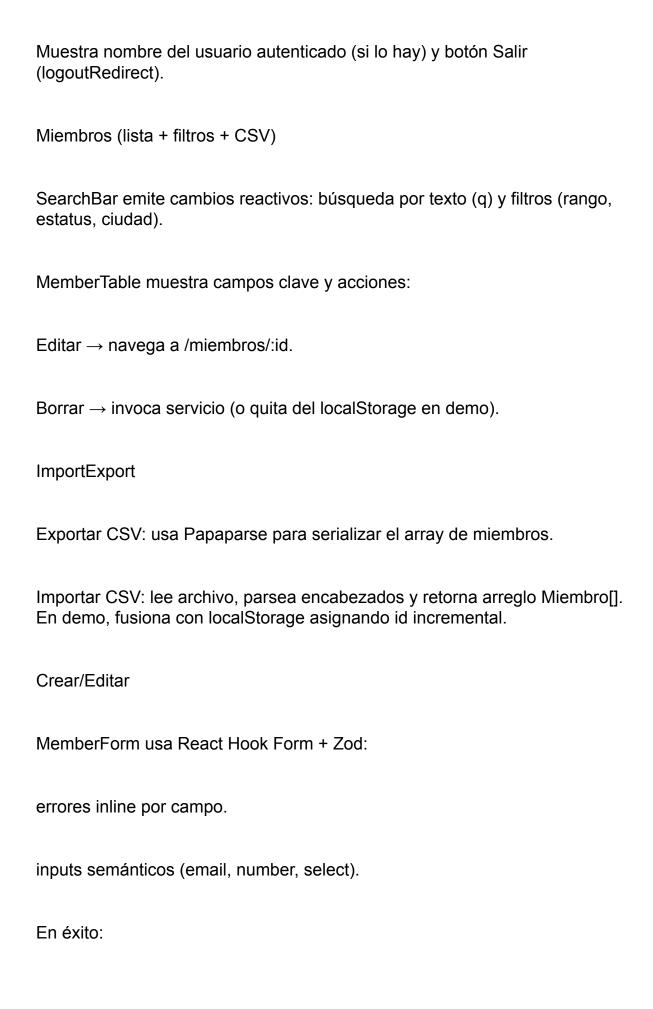
Crear/Editar/Borrar muta localStorage y actualiza estado.

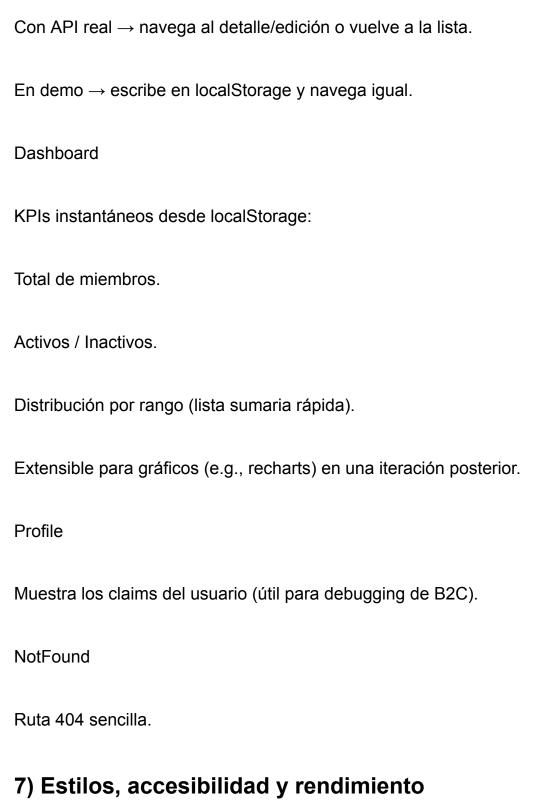
Esto permite demo end-to-end en clase.

6) UI y flujo funcional

Navbar

Link a Miembros y Dashboard.





Tailwind para composición rápida y consistente.

Accesibilidad (A11y): labels asociadas a inputs, elementos nativos (button, select), contraste base Tailwind. Se puede reforzar con:

foco visible (focus:ring), mensajes aria-live para errores, y toasts accesibles.

Rendimiento:

Vite + React 18, división por rutas trivial (router listo).

Estado local por página; sin global store innecesario.

Memo en filtros (useMemo) para evitar recomputar en cada render.

8) Variables de entorno y ejecución

.env.example

VITE API BASE URL=http://localhost:5000

VITE_B2C_CLIENT_ID=REEMPLAZA

VITE_B2C_AUTHORITY=https://TENANT.b2clogin.com/TENANT.onmicrosoft.com/B2C_1_signupsignin

VITE_B2C_KNOWN_AUTHORITY=TENANT.b2clogin.com

Comandos

npm i

npm run dev # http://localhost:5173

npm run build #/dist listo para hosting estático

npm run preview # sirve el build localmente

Cambiar de demo a API real

Asegura que tu backend expone los endpoints REST con CORS habilitado.

Edita VITE_API_BASE_URL para apuntar a tu API.

Con B2C activo, el interceptor Axios añadirá Bearer a cada request.

9) Manejo de errores y edge cases

Auth: si no hay sesión, PrivateRoute dispara login. Si B2C no está configurado aún, verás redirecciones al dominio de B2C no resueltas: trabaja en demo hasta completar B2C.

Servicios REST: cualquier excepción de listMembers/getMember en clase cae al demo (semilla + localStorage) para no bloquear la práctica.

CSV: encabezados deben coincidir con los nombres de las propiedades; el import es tolerante, pero conviene limpiar CSV (e.g., cilindrajeCC numérico).

IDs: en demo se autoincrementan; con API real, el backend define el id.

10) Testing manual sugerido (en clase)

Lista \rightarrow crear \rightarrow editar \rightarrow borrar en demo.

Filtros combinados: q + rango + estatus + ciudad.

Exportar CSV, limpiar tabla, reimportar y verificar consistencia. Cambiar .env.local a un backend de prueba y repetir CRUD. Probar loginRedirect y logoutRedirect (cuando ya tengas B2C). 11) Despliegue recomendado Frontend: Azure Static Web Apps o Netlify/Vercel (sólo subir /dist). Configura rutas SPA (fallback a index.html). Backend: Azure App Service o Azure Functions (con API REST). Habilita CORS y validación de JWT de B2C. Variables: Cargar VITE_* en el entorno del host estático. 12) Extensiones sugeridas (futuro inmediato) Backend ASP.NET Core + EF Core (Azure SQL) con:

Entidad Miembro, DbContext, migraciones.

Controlador MiembrosController y DTOs.

Bearer JWT validado contra B2C.

Roles/Scopes: autorización por rango o claims B2C (e.g., sólo "Admin" puede borrar).

Toasts (éxito/error) y paginación en la tabla.

Gráficas en Dashboard (recharts) y KPIs por ciudad/estatus.

i18n (react-intl) si habrá múltiples idiomas.

E2E tests (Playwright) del flujo CRUD y filtros.

Checklist de cumplimiento del reto

- React + TypeScript (Opción 3).
- Autenticación social vía Azure AD B2C con MSAL listo.
- ✓ CRUD completo de miembros (UI + servicios + validación).
- Búsqueda y filtros por campos clave.
- ✓ Importación/Exportación CSV.
- Dashboard con métricas.

- Arquitectura limpia con separación de responsabilidades.
- ✓ Modo demo para trabajar en clase sin backend.
- ✓ Listo para conectar a API real y desplegar.