¡Excelente idea! Preparar un archivo README.md es una práctica fundamental en el desarrollo de software. Nos servirá como una "hoja de ruta" para entender rápidamente el proyecto y facilitará mucho nuestras futuras colaboraciones.

Aquí te he preparado un README.md completo, basado en todo lo que hemos construido y discutido. Podés guardar este texto en un archivo llamado README.md en la raíz de tu proyecto.

**CRM Financiero**

Este es un CRM (Customer Relationship Management) personalizado, construido como una Single Page Application (SPA) con React, diseñado para la gestión de clientes y oportunidades de negocio en el sector financiero. El corazón de la aplicación es un embudo de negocios interactivo estilo Kanban que permite un seguimiento visual del progreso de cada oportunidad.

**✨ Características Principales**

* **Gestión de Clientes:** Permite crear, editar y visualizar un listado completo de clientes, tanto personas físicas como jurídicas.
* **Embudo de Negocios Interactivo:** Una vista Kanban (FunnelView) donde las oportunidades de negocio (negocios) se pueden arrastrar y soltar entre distintas etapas.
* **Gestión de Negocios:** Soporta la creación de múltiples negocios por cliente y permite modificar el estado de un negocio desde la vista de detalle del cliente.
* **Consulta de Deudores:** Integra una pestaña para consultar la situación crediticia de un cliente, con un fallback a datos de ejemplo para desarrollo.
* **Persistencia de Datos:** Guarda automáticamente todo el estado de la aplicación (clientes, negocios, etc.) en el localStorage del navegador.
* **Importación y Exportación:** Incluye funciones para guardar un backup completo del estado de la aplicación en un archivo .json y para restaurarlo.

**💻 Stack Tecnológico**

* **Frontend:** [React.js](https://reactjs.org/)
* **Estilos:** [Tailwind CSS](https://tailwindcss.com/)
* **Íconos:** [Lucide React](https://lucide.dev/)
* **Drag and Drop:** [React Beautiful DnD](https://github.com/atlassian/react-beautiful-dnd)
* **Estado:** Hooks de React (useState, useEffect, useMemo) a nivel de componente (App.js).

**📁 Estructura del Proyecto**

La estructura de carpetas está organizada por funcionalidad para mantener el código ordenado y escalable.

/src

├── components/

│ ├── clients/ # Componentes específicos de la vista de clientes (Detail, Form, List)

│ ├── common/ # Componentes reutilizables (Accordion, InfoItem, InputField)

│ ├── funnel/ # Componentes del embudo (Column, Card)

│ ├── modals/ # Modales para acciones específicas (FunnelStatus, Qualification, etc.)

│ └── tabs/ # Componentes para las pestañas de la vista de detalle (Summary, DebtorStatus, etc.)

│

├── hooks/

│ └── useFunnel.js # Hook personalizado con la lógica del embudo de negocios

│

├── views/

│ ├── App.js # Componente raíz, cerebro de la aplicación

│ ├── ClientsView.js # Vista principal de gestión de clientes

│ ├── FunnelView.js # Vista principal del embudo de negocios

│ └── ... # Otras vistas principales (Dashboard, SGR, etc.)

│

├── data.js # Carga y exporta los datos iniciales (lee el .json)

├── crm\_negocios.json # Archivo de base de datos en formato JSON

└── index.js # Punto de entrada de la aplicación React

**🧠 Lógica y Arquitectura**

La aplicación sigue varios principios clave de diseño:

**1. Estado Centralizado en App.js**

App.js actúa como el **cerebro** de la aplicación. Mantiene el estado principal de todos los datos (listas de clientes, negocios, SGRs, etc.) usando el hook useState. Toda la información fluye desde App.js hacia abajo a los componentes hijos a través de props.

**2. Modelo de Datos Dual: Clientes y Negocios**

La aplicación maneja dos listas de datos principales que están relacionadas:

* **clients**: Una lista de todos los clientes. Cada objeto contiene la información de contacto y detalles del cliente.
* **negocios**: Una lista de todas las oportunidades de negocio. Cada objeto negocio contiene:
  + Detalles de la oportunidad (nombre, monto, etc.).
  + La propiedad **estado**, que define su etapa en el embudo.
  + Un objeto **cliente anidado** con la información básica del cliente al que pertenece.

Esta estructura permite que un solo cliente pueda tener múltiples negocios asociados.

**3. Flujo de Datos y Funciones "Handler"**

Para modificar el estado, los componentes hijos (como ClientsView o FunnelView) no lo hacen directamente. En su lugar, App.js les pasa funciones "handler" como props (ej: onUpdateNegocio, onAddClientAndBusiness). Cuando un componente hijo necesita cambiar algo, llama a una de estas funciones, y App.js se encarga de actualizar su propio estado. Esto asegura un flujo de datos predecible y unidireccional.

**4. Persistencia en localStorage**

Un useEffect en App.js vigila cualquier cambio en los datos principales (clients, negocios, etc.). Cuando detecta un cambio, guarda una copia completa del estado en el localStorage del navegador. Al recargar la página, otro useEffect intenta leer estos datos guardados para restaurar la aplicación a su último estado.

**🚀 Próximas Mejoras**

Para futuras versiones, podemos enfocarnos en:

* Implementar completamente las funciones pendientes (ej: onDeleteClient, onAddNewBusiness para múltiples negocios en la vista de detalle).
* Desarrollar un backend real (ej: con Node.js/Express y una base de datos como PostgreSQL) para reemplazar localStorage y permitir el uso multiusuario.
* Crear un "proxy" en el backend para hacer las llamadas a la API del BCRA de forma segura y evitar las restricciones CORS.
* Mejorar la vista de detalle del cliente para que pueda mostrar y gestionar una lista de múltiples negocios activos.