ROADMAP DESARROLLO

🏗️ FASE 1: Data Engine & Advanced Grid (Especificación Técnica)

Objetivo del Sprint: Construir el gestor de proyectos, el motor de ingesta de datos (SABI) y la interfaz de tabla interactiva (Grid) que permita organizar, personalizar y mover datos sin intervención de IA todavía.

Stack: Python 3.10+, Streamlit, Pandas, AgGrid (streamlit-aggrid).

---

1. ARQUITECTURA DE DATOS (BACKEND)

Para cumplir el requisito de **"Mover filas entre tablas"** de forma eficiente, no usaremos archivos separados. Usaremos un **Master DataFrame** con una columna de control.

1.1 Estructura de Archivos

El sistema debe gestionar los datos localmente en esta estructura:

Plaintext

```
/projects_data
  /Sector_Vitivinicola_2025/
      ├── master_data.parquet   # (Base de Datos: Todas las empresas + columnas custom)
      └── schema_config.json    # (Metadatos: Definición de vistas, etiquetas y opciones)
```

1.2 Esquema del `master_data.parquet`

Al cargar el Excel de SABI, el backend debe normalizarlo y añadir columnas de sistema:

Columna	Tipo	Origen	Descripción
`_uid`	string	Sistema	ID único (UUID) para gestionar movimientos.
`_list_id`	string	Sistema	Indica en qué tabla está (`inbox` , `shortlist` , `trash`).
`name`	string	SABI	Nombre de la empresa.

`revenue`	float	SABI	Ventas normalizadas.
`ebitda`	float	SABI	EBITDA normalizado.
`...`	...	SABI	Resto de columnas originales.
`[Tag] Estado`	string	Custom	Columna creada por usuario (inicialmente vacía).

1.3 Esquema del `schema_config.json`

Controla la configuración de la UI.

JSON

```
{
 "lists": [
   {"id": "inbox", "name": "📥 Bandeja de Entrada"},
   {"id": "shortlist", "name": "⭐ Shortlist"},
   {"id": "discarded", "name": "🗑️ Descartados"}
 ],
 "custom_columns_definitions": {
   "status": {
      "type": "single_select",
      "options": ["Nuevo", "Contactado", "Reunión", "NDA"]
   }
 }
}
```

---

2. PANTALLA 1: LANDING & PROYECTOS

Funcionalidad: Gestión de espacios de trabajo.

1. **Sidebar:** Muestra lista de carpetas encontradas en `/projects_data`.
2. **Área Principal:**
   - **"Nuevo Proyecto":** Input text + Botón "Crear".
     - *Acción:* Crea la carpeta y un `schema_config.json` por defecto.
   - **"Cargar Proyecto":** Al seleccionar uno de la sidebar, carga el `master_data.parquet` en `st.session_state['df_master']`.

---

3. PANTALLA 2: INGESTA (UPLOADER)

Funcionalidad: Convertir el Excel sucio de SABI en el formato Parquet limpio.

1. **Componente:** `st.file_uploader` (acepta .xlsx, .csv).
2. **Lógica de Normalización (ETL):**
   - Detectar cabeceras. SABI suele usar nombres largos. Mapear:
     - `Ingresos de explotación...` → `revenue`
     - `Resultado del Ejercicio...` → `net_income`
   - Generar `_uid` para cada fila.
   - Asignar `_list_id = 'inbox'` a todas las filas.
3. **Persistencia:** Guardar inmediatamente como `master_data.parquet`.

---

4. PANTALLA 3: THE ADVANCED GRID (NÚCLEO)

Esta es la parte crítica. La UI debe renderizar **una vista filtrada** del Master Data según la pestaña seleccionada.

4.1 Navegación (Tabs)

- Renderizar pestañas superiores iterando sobre `schema_config['lists']`.
- Al hacer clic en "Shortlist", filtrar el dataframe:

  df_view = st.session_state['df_master'][st.session_state['df_master']['_list_id'] == 'shortlist']

4.2 Configuración de AgGrid (Código Crítico)

Para permitir **mover columnas, ocultarlas y seleccionar filas**, usaremos esta configuración específica:

Python

```
from st_aggrid import AgGrid, GridOptionsBuilder, GridUpdateMode

def render_grid(df_view, custom_defs):
    gb = GridOptionsBuilder.from_dataframe(df_view)

    # 1. Habilitar Selección y Checkboxes
    gb.configure_selection(selection_mode="multiple", use_checkbox=True)

    # 2. Habilitar Panel Lateral (SideBar) para mover/ocultar columnas
    gb.configure_side_bar()
```

```
    # 3. Configurar Columnas Custom (Etiquetas)
    # Si detectamos una columna definida como "single_select" en el JSON
    for col_name, config in custom_defs.items():
        if config['type'] == 'single_select':
            gb.configure_column(
                col_name,
                editable=True,
                cellEditor='agSelectCellEditor',
                cellEditorParams={'values': config['options']}
            )

    # 4. Ocultar columnas de sistema
    gb.configure_column("_uid", hide=True)
    gb.configure_column("_list_id", hide=True)

    gridOptions = gb.build()

    response = AgGrid(
        df_view,
        gridOptions=gridOptions,
        enable_enterprise_modules=True, # Necesario para SideBar completa
        update_mode=GridUpdateMode.MODEL_CHANGED
    )
    return response
```

---

5. INTERACCIONES: AÑADIR COLUMNAS Y MOVER FILAS

5.1 Añadir Nueva Columna (Modal)

Botón en Toolbar `[+ Columna]`. Abre un `st.form`:

- **Input:** Nombre (ej. "Prioridad").
- **Radio:** Tipo ("Texto Libre" vs "Etiqueta").
- **Si es Etiqueta:** Input text para opciones ("Alta,Media,Baja").

Acción Backend:

1. Añadir columna al `df_master` llena con `None` (o valor por defecto).
2. Si es Etiqueta, actualizar `schema_config.json` con las opciones.
3. Guardar a disco y recargar (`st.rerun`).

5.2 Mover Filas (Action Bar)

Si el usuario selecciona filas en el Grid, aparece una barra flotante debajo:

- **UI:** `st.container` condicional (si `len(selection) > 0`).
- **Dropdown:** "Mover a..." (Lista de tablas disponibles: Inbox, Shortlist, Trash).
- **Botón:** "Confirmar Movimiento".

Lógica de Movimiento:

Python

```
def move_rows(selected_uids, target_list_id):
    # Cargamos el maestro
    df = st.session_state['df_master']

    # Actualizamos solo la columna de control _list_id
    df.loc[df['_uid'].isin(selected_uids), '_list_id'] = target_list_id

    # Guardamos y recargamos
    save_parquet(df)
    st.session_state['df_master'] = df
    st.rerun()
```

---

6. CREACIÓN DE NUEVAS TABLAS (LISTAS)

Botón `[+ Nueva Lista]` al lado de los Tabs.

- **Acción:** Añade una entrada a `schema_config['lists']` (ej. `{"id": "watchlist", "name": "👀 Watchlist"}`).
- **Resultado:** Aparece una nueva pestaña vacía. El usuario ahora puede seleccionar filas de "Inbox" y moverlas a "Watchlist".

RESUMEN DE ENTREGABLES FASE 1

1. **`data_manager.py`**: Funciones `load_project`, `save_project`, `normalize_sabi`.
2. **`app.py`**:
    - Sidebar de gestión de proyectos.
    - Lógica de Tabs (Inbox/Shortlist...).
    - Lógica de "Mover Filas" (actualizar `_list_id`).

3. **Componente AgGrid**: Configurado con `sideBar=True` para permitir al usuario ocultar/mostrar cualquier columna del Excel original.

 TICKET-01: Configuración del Proyecto y Data Manager (Backend)

Tipo: Backend / Core
Prioridad: Alta

Descripción:
Crear la estructura base del repositorio y la lógica de persistencia de datos. Necesitamos una clase o módulo que gestione la lectura y escritura de los archivos Parquet y JSON de configuración. No usaremos base de datos SQL, sino sistema de archivos local.

Requerimientos Técnicos:

1. Crear estructura de directorios: `/projects_data/{project_id}/`.
2. Implementar `data_manager.py` con las siguientes funciones:
   - `create_project(name)`: Crea la carpeta y un `schema_config.json` por defecto.
   - `save_master_data(project_id, df)`: Guarda el DataFrame en formato `.parquet` (usar motor `pyarrow`).
   - `load_master_data(project_id)`: Devuelve el DataFrame leyendo el `.parquet`.
   - `update_schema(project_id, new_config)`: Guarda cambios en el JSON de configuración.
3. Definir el `schema_config.json` inicial con la estructura de listas: `inbox`, `shortlist`, `discarded`.

Criterios de Aceptación:

- [ ] Se pueden crear carpetas de proyectos programáticamente.

☐ Se puede guardar un DataFrame de Pandas y recuperarlo intacto (manteniendo tipos de datos float/int/str) usando Parquet.

TICKET-02: Interfaz de Gestión de Proyectos (Landing)

Tipo: Frontend / UI
Prioridad: Alta

Descripción:
Implementar la primera pantalla que ve el usuario al entrar en la aplicación (`app.py`). Debe permitir crear un nuevo espacio de trabajo o cargar uno existente.

Requerimientos Técnicos:

1. **Sidebar de Streamlit:**
   - Listar carpetas existentes en `/projects_data`.
   - Al hacer clic en uno, cargar el `project_id` en `st.session_state`.
2. **Área Principal (Si no hay proyecto seleccionado):**
   - Input Text: "Nombre del Nuevo Proyecto".
   - Botón: "Crear".
   - Al crear, debe llamar a `data_manager.create_project` y redirigir a la vista del proyecto.

Criterios de Aceptación:

☐ El usuario ve la lista de proyectos guardados.

☐ El usuario puede crear "Sector X" y esto genera la carpeta correspondiente.

☐ Al seleccionar un proyecto, la interfaz cambia para mostrar el título del proyecto activo.

TICKET-03: Motor de Ingesta y Normalización SABI (ETL)

Tipo: Backend / Data Processing
Prioridad: Alta

Descripción:
Crear la lógica para procesar el Excel "sucio" que viene de SABI. Debemos

renombrar columnas complejas a nombres internos simples y añadir identificadores de sistema.

Requerimientos Técnicos:

1. Implementar función `normalize_sabi_data(uploaded_file)`:
    - Leer Excel/CSV con Pandas.
    - **Renombrado (Mapping):**
        - `Nombre` → `name`
        - `Dirección web` → `website`
        - `Código NIF` → `nif`
        - `Ingresos de explotación...` → `revenue`
        - `EBITDA...` → `ebitda`
        - `Resultado del Ejercicio...` → `net_income`
        - `Localidad` → `city`
        - `Descripción actividad` → `description`
    - **Columnas de Sistema:**
        - Generar `_uid`: UUID único para cada fila.
        - Generar `_list_id`: Valor por defecto `'inbox'`.
2. Integrar en UI: Añadir `st.file_uploader` en la pantalla principal si el proyecto está vacío.

Criterios de Aceptación:

- [ ] El sistema acepta un archivo `.xlsx` de SABI real.
- [ ] El DataFrame resultante tiene las columnas renombradas correctamente (ej. `ebitda` en lugar de la frase larga).
- [ ] Cada fila tiene un `_uid` único.
- [ ] El archivo se guarda automáticamente como `master_data.parquet` tras la subida.

TICKET-04: Implementación de AgGrid Básico y Sidebar

Tipo: Frontend / Grid
Prioridad: Crítica

Descripción:
Renderizar la tabla principal usando `st_aggrid`. Debe ser capaz de manejar miles de filas sin congelar el navegador y permitir la manipulación de columnas.

Requerimientos Técnicos:

1. Configurar `GridOptionsBuilder`:
    - `pagination=True`, `paginationPageSize=50`.
    - `rowSelection='multiple'` (con checkboxes).
    - `sideBar=True` (Habilitar panel lateral nativo de AgGrid para ocultar/mostrar columnas).
2. **Configuración de Columnas:**
    - `name`: Pinned Left (Fija a la izquierda).
    - `revenue`, `ebitda`: Formato numérico compactado (ej. usando `valueFormatter` JS).
    - Ocultar columnas de sistema (`_uid`, `_list_id`) por defecto (`hide=True`).
3. Renderizar el Grid pasando el DataFrame del proyecto activo.

Criterios de Aceptación:

- [ ] La tabla carga los datos del Parquet.
- [ ] Se pueden seleccionar filas con checkboxes.
- [ ] Existe un panel lateral donde el usuario puede arrastrar columnas para reordenarlas u ocultarlas.
- [ ] Los números se ven formateados (no `1500000` sino `1.5M`).

 TICKET-05: Navegación por Pestañas (Filtrado de Vistas)

Tipo: Frontend / Logic
Prioridad: Media

Descripción:
Implementar la lógica para ver diferentes "listas" (Inbox, Shortlist, Descartados)

usando pestañas, sin duplicar datos.

Requerimientos Técnicos:

1. Leer las listas definidas en `schema_config.json`.
2. Crear Tabs en Streamlit (`st.tabs`) iterando sobre esas listas.
3. **Lógica de Filtrado:**
    - Cuando el usuario está en el Tab "Shortlist", filtrar el DataFrame maestro antes de pasarlo al AgGrid:
    - `df_view = df_master[df_master['_list_id'] == 'shortlist']`

Criterios de Aceptación:

- [ ] Al cambiar de pestaña, la tabla muestra solo las empresas correspondientes a esa lista.
- [ ] Las empresas recién subidas aparecen solo en "Inbox".

TICKET-06: Acciones en Lote (Mover Filas)

Tipo: Fullstack
Prioridad: Alta

Descripción:
Permitir al usuario seleccionar filas y moverlas a otra lista (ej. de Inbox a Shortlist).

Requerimientos Técnicos:

1. **Frontend:** Detectar selección de filas en AgGrid (`grid_response['selected_rows']`).
2. **UI Flotante:** Si hay selección, mostrar `st.container` debajo de la tabla con:
    - Selectbox: "Mover a..." (Lista de tablas disponibles).
    - Botón: "Confirmar".
3. **Backend Logic:**
    - Función `move_rows(uids, target_list_id)`.
    - Debe cargar el `master_data.parquet`.
    - Actualizar la columna `_list_id` para los UIDs coincidentes.

- Guardar Parquet y forzar `st.rerun()`.

Criterios de Aceptación:

- [ ] Seleccionar 3 empresas en "Inbox", moverlas a "Shortlist".
- [ ] Al ir a la pestaña "Shortlist", las empresas están ahí.
- [ ] Al volver a "Inbox", ya no están.

🎟️ TICKET-07: Gestión de Columnas Personalizadas (Manuales)

Tipo: Fullstack
Prioridad: Media

Descripción:
Permitir al usuario añadir nuevas columnas (Tags o Texto) que se persistan en el esquema y en el dataframe.

Requerimientos Técnicos:

1. **UI Modal:** Botón `[+ Columna]` que abre un formulario.
   - Inputs: Nombre columna, Tipo (Texto/Etiqueta), Opciones (si es etiqueta).
2. **Backend:**
   - Actualizar `master_data.parquet` añadiendo la columna nueva (inicializada a `None` o `""`).
   - Actualizar `schema_config.json` con la definición de la columna (necesario para saber si es un dropdown).
3. **AgGrid Update:**
   - Si la columna es tipo "Etiqueta", configurar `cellEditor='agSelectCellEditor'` con las opciones guardadas en el JSON.
   - Hacer la columna `editable=True`.

Criterios de Aceptación:

- [ ] Usuario crea columna "Estado" con opciones "Pendiente, Contactado".
- [ ] La columna aparece en la tabla.

- [ ] El usuario puede hacer doble clic en una celda de "Estado" y elegir "Contactado" de un dropdown.
- [ ] Al recargar la página, el valor "Contactado" persiste.