This is a copy of a conversation between ChatGPT & Anonymous.
Report conversation

Lo único que sería algo que documentar y capacitarlos en el momento es que en el detalle del cliente ponen el DNI, ROOT, etc., y lo ponen completo, tal cual sale en el documento. No como antes lo hacían, que era un subsegmento que hacían por un tema de la herramienta de antes. Ahora sí lo ponen tal cual y nosotros por detrás lo procesamos. Eso y que se llene todo menos lo del flag y lo del accionado, porque todo lo demás se puede jalar de toda la información. Cuando mantienes el hover, creo que se achica para mostrar que está siendo seleccionado el botón y para achicarse sale del espacio del mouse. Porque se achica y el mouse ya no está dentro del nuevo campo de hover, entonces se vuelve a agrandar y se vuelve a achicar, y ese es otro problema. Mejor quitar la animación de achicado de los botones, que los botones cambian de color pero no se achican. Después productos... Bueno, lo que te voy a pedir es, tal cual investigadores, están viendo para el lunes en la tarde, los scripts, algo de productos hay que abrazarlos, por si nos han pedido. ¿Qué productos vamos a chepar? Sí, principalmente acá vamos a poner cuentos y tarjetas. Sí, pero eso es lo que más... Las cuentas de préstamos... Y no es todos los campos, es que necesitamos bajar todos los campos. El tipo de whatever... Ajá, el tipo y el monto desembolsado. Y el cliente solo para hacer la asociación. El check de validación de cliente que ha registrado y se ha asociado, es el asociado con la vaina. Yo por mi cuenta este fin de semana tengo que leer sobre encriptado, in place, encriptado en movimiento. Sí, porque después vamos a trabajar con eso mismo. ¿Qué más jalaría? Si vamos a las tablas... ¿Qué pasó? Lo rompí sin querer. Ayuda... Ya no sé qué hice, puse monto investigador y eso se dice. No, fuck. A ver, en el producto... No, se re rompió. Ya, puedes hacer clic en el monto investigador para ver si se puede replicar haciendo solo clic en el monto investigador. Ah, si haces clic y el cursor entra en el monto investigador, desaparece. Ya, elimina el producto y cierra el app y vuelve a abrirlo. Mejor para solo chequear que no es un tema de que está fregado el... A ver, abre por favor. A ver, elimina productos. Ya no hay nada, ¿no? ¿Ya no hay nada? ¿No hay ningún producto? Es que cuando pongo eliminar, esto se queda como producto 1 y después sale el otro. Ah, y sale bien abajo, ¿no? Es como si todo el espacio de formulario se quedara ahí pegado. Y haz clic en otro campo de Monteperde, por ejemplo. Quizá tengamos eso para cada... Algo con la validación de campos. En la primera fila al parecer nada más porque... La primera columna, la primera columna, sí. Ha desaparecido, sí. ¿Se han movido las alertas? No. Algo ha pasado, sí. Esas alertas deberían estar a ese nivel, esas alertas deberían estar a ese nivel. Algo con ese movimiento parece ser. Sí, a ver, elimina y vuelve a crear. Que tapa las... Tapa las... Las como, las... Las texturas. Nada más. Haz clic en Promonto de acá. Y así están todos. Y sí, igual le haces clic en Monteperde Proceso que está en la segunda columna y desaparece también. Todos los campos de la primera columna del registro de producto, ¿no? Para mí es algo que tiene... Para mí es un overflow, algo. Sí, sí, sí, algo está pasando que se está chancando. Está saliendo del margen. También quiero ver si... Parece que cuando, mira, hay un flag de... Y ahí dice producto... No, la ID... Llenas la ID que también está en la primera columna y no pasa nada. Baja. Es que creo que, mira, si seleccionas Monte Investigado de nuevo... Hay un... Este... Este tooltip de ingresa ambas fechas para variar la secuencia. A ver, ingresa ambas fechas. A ver si es que eso... Te quita... O sea, si ingresas ambas fechas... Quizá eso regresa al... No es un... Ah, tengo que mirar porque técnicamente había puesto un timer para que no te saliera cada 30 minutos el nuevo llave técnica. Bien, el producto, ajá. A ver, hazle dos fechas. A ver. Ponga la ID y ponga una fecha y pon la otra fecha. Para ver si es que si al poner las fechas dos... Se satisface la validación y desaparece... Ya. Ajá. Progreso bien. Ajá.

Fecha de currencia ponga ahí. Sí, no te deja porque se abre y ni siquiera se te abre... Se abre, debe de ingresar la fecha de ocurrencia y de descubrimiento y desaparece la primera columna de... Buenas. Regreso... Ahí ya tenemos un par de bugs.

Thought for 50s

Aquí hay una primera versión completa del front-end web app incorporando lo que contaste en el audio/texto:

- **DNI/RUC se capturan completos** (sin subsegmentar).
- **Los botones YA NO se achican en hover**: solo cambian de color (adiós bug de "flicker" por hover).
- **Secciones dinámicas** de Clientes, Colaboradores e Productos.
- **Validaciones** (incluye fechas de ocurrencia/descubrimiento, montos numéricos, unicidad de IDs, etc.).
- **Carga masiva por CSV** para clientes, colaboradores y productos (con manejo de duplicados).
- **Auto-save** con `localStorage` en cada cambio y restauración automática al cargar.
- **Checkpoints** (snapshots de todo el caso) que puedes guardar y restaurar.
- **Generación de reporte .doc** (HTML dentro de un `.doc` que Word puede abrir).
- **Tabla resumen** de todos los registros (clientes, colaboradores, productos) para ver rápidamente lo que está cargado.
- Sin animación de "shrink" en botones, solo color; no hay transforms que cambien el tamaño del botón en hover.

`index.html`

html Copy code

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Gestor de Casos de Fraude</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="style.css">
</head>
<body>
<header class="app-header">
    <h1>Gestor de Casos de Fraude</h1>
    <p class="subtitle">
        Prototipo web — Registro de clientes, colaboradores y productos
    </p>
</header>

<main class="app-main">
    <!-- Información general del caso -->
    <section class="card" id="case-info-section">
        <h2>Información del caso</h2>
        <div class="grid-2">
            <div class="form-group">
                <label for="caseId">ID del caso</label>
                <input type="text" id="caseId" placeholder="Ej: N2025-00
            </div>
            <div class="form-group">
                <label for="caseTitle">Título / Nombre del caso</label>
                <input type="text" id="caseTitle" placeholder="Ej: Uso i
            </div>
            <div class="form-group">
                <label for="leadInvestigator">Investigador responsable</
                <input type="text" id="leadInvestigator" placeholder="No
            </div>
            <div class="form-group">
                <label for="caseStartDate">Fecha de inicio</label>
                <input type="date" id="caseStartDate">
            </div>
        </div>
        <div class="form-group">
            <label for="caseSummary">Resumen breve del caso</label>
            <textarea id="caseSummary" rows="3"
                      placeholder="Descripción corta de los antecedentes
        </div>
    </section>
```

```html
<!-- CLIENTES -->
<section class="card" id="clients-section">
    <div class="section-header">
        <h2>Clientes</h2>
        <div class="section-actions">
            <label class="file-label">
                Carga masiva CSV
                <input type="file" id="clientsCsvInput" accept=".csv
            </label>
            <button type="button" class="btn" id="addClientBtn">+ Añ
        </div>
    </div>
    <p class="helper-text">
        Ingresar el documento completo (DNI, RUC, CE, etc.) tal como
        El sistema de backend se encargará de procesarlo (ya no se u
    </p>
    <div id="clientsContainer" class="instance-container">
        <!-- Filas de clientes dinámicas -->
    </div>
</section>

<!-- COLABORADORES / INVESTIGADORES -->
<section class="card" id="team-section">
    <div class="section-header">
        <h2>Colaboradores / Investigadores</h2>
        <div class="section-actions">
            <label class="file-label">
                Carga masiva CSV
                <input type="file" id="teamCsvInput" accept=".csv">
            </label>
            <button type="button" class="btn" id="addTeamMemberBtn">
        </div>
    </div>
    <div id="teamContainer" class="instance-container">
        <!-- Filas de colaboradores dinámicas -->
    </div>
</section>

<!-- PRODUCTOS -->
<section class="card" id="products-section">
```

```
            <div class="section-header">
                <h2>Productos</h2>
                <div class="section-actions">
                    <label class="file-label">
                        Carga masiva CSV
                        <input type="file" id="productsCsvInput" accept=".cs
                    </label>
                    <button type="button" class="btn" id="addProductBtn">+ A
                </div>
            </div>
            <p class="helper-text">
                Campos como <strong>flag</strong> y <strong>accionado</stron
                el resto de campos se usan para jalar información y validar
            </p>
            <div id="productsContainer" class="instance-container">
                <!-- Filas de productos dinámicas -->
            </div>
        </section>

        <!-- RESUMEN DE DATOS -->
        <section class="card" id="summary-section">
            <h2>Resumen de datos</h2>
            <p class="helper-text">
                Vista compacta de todos los clientes, colaboradores y produc
            </p>
            <div id="summaryTablesWrapper" class="summary-wrapper">
                <div class="summary-block">
                    <h3>Clientes</h3>
                    <table id="summaryClientsTable">
                        <thead>
                        <tr>
                            <th>ID cliente</th>
                            <th>Nombre</th>
                            <th>Tipo doc.</th>
                            <th>N° documento</th>
                        </tr>
                        </thead>
                        <tbody></tbody>
                    </table>
                </div>
                <div class="summary-block">
                    <h3>Colaboradores</h3>
                    <table id="summaryTeamTable">
                        <thead>
                        <tr>
                            <th>ID colaborador</th>
                            <th>Nombre</th>
                            <th>Rol</th>
                        </tr>
                        </thead>
                        <tbody></tbody>
                    </table>
                </div>
                <div class="summary-block">
                    <h3>Productos</h3>
                    <table id="summaryProductsTable">
                        <thead>
                        <tr>
                            <th>ID producto</th>
                            <th>Tipo</th>
                            <th>Cliente asociado</th>
                            <th>Monto desembolsado</th>
                            <th>Monto investigado</th>
                            <th>Monto pérdida</th>
                        </tr>
                        </thead>
                        <tbody></tbody>
                    </table>
                </div>
            </div>
        </section>

        <!-- CONTROLES GENERALES -->
        <section class="card" id="controls-section">
            <h2>Guardado y reportes</h2>
            <div class="controls-grid">
                <div class="form-group">
                    <label for="checkpointSelect">Checkpoints guardados</lab
                    <select id="checkpointSelect">
                        <option value="">(Ningún checkpoint seleccionado)</o
```

```html
            </select>
        </div>
        <div class="button-group-vertical">
            <button type="button" class="btn secondary" id="saveChec
                Guardar checkpoint actual
            </button>
            <button type="button" class="btn secondary" id="restoreC
                Restaurar checkpoint seleccionado
            </button>
            <button type="button" class="btn danger" id="clearAllBtn
                Limpiar todo el formulario
            </button>
        </div>
        <div class="button-group-vertical">
            <button type="button" class="btn primary" id="generateRe
                Generar reporte (.doc)
            </button>
            <small class="helper-text">
                El archivo incluirá el ID del caso en el nombre.
            </small>
        </div>
    </div>
    <div id="globalErrorBox" class="global-error" hidden></div>
    <div id="globalInfoBox" class="global-info" hidden></div>
</section>

<!-- Plantillas para elementos dinámicos -->
<template id="clientRowTemplate">
    <div class="instance-row client-row">
        <div class="row-header">
            <h3>Cliente <span class="row-index"></span></h3>
            <button type="button" class="icon-btn remove-btn" title=
        </div>
        <div class="grid-3">
            <div class="form-group">
                <label>ID cliente*</label>
                <input type="text" class="client-id-input" required>
                <div class="error-message"></div>
            </div>
            <div class="form-group">
```

```
                    <label>Nombre completo*</label>
                    <input type="text" class="client-name-input" require
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Tipo documento*</label>
                    <select class="client-doc-type-input" required>
                        <option value="">Seleccionar...</option>
                        <option value="DNI">DNI</option>
                        <option value="RUC">RUC</option>
                        <option value="CE">Carné de extranjería</option>
                        <option value="PAS">Pasaporte</option>
                    </select>
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Número de documento*</label>
                    <input type="text" class="client-doc-number-input" r
                           placeholder="Tal cual figura en el documento"
                    <div class="error-message"></div>
                </div>
                <div class="form-group full-width">
                    <label>Notas / comentarios</label>
                    <input type="text" class="client-notes-input">
                </div>
            </div>
        </div>
    </template>

    <template id="teamRowTemplate">
        <div class="instance-row team-row">
            <div class="row-header">
                <h3>Colaborador <span class="row-index"></span></h3>
                <button type="button" class="icon-btn remove-btn" title=
            </div>
            <div class="grid-3">
                <div class="form-group">
                    <label>ID colaborador*</label>
                    <input type="text" class="team-id-input" required>
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Nombre completo*</label>
                    <input type="text" class="team-name-input" required>
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Rol / Función</label>
                    <input type="text" class="team-role-input" placehold
                </div>
            </div>
        </div>
    </template>

    <template id="productRowTemplate">
        <div class="instance-row product-row">
            <div class="row-header">
                <h3>Producto <span class="row-index"></span></h3>
                <button type="button" class="icon-btn remove-btn" title=
            </div>
            <div class="grid-3">
                <div class="form-group">
                    <label>ID producto*</label>
                    <input type="text" class="product-id-input" required
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Tipo de producto*</label>
                    <select class="product-type-input" required>
                        <option value="">Seleccionar...</option>
                        <option value="Cuenta">Cuenta</option>
                        <option value="Tarjeta">Tarjeta</option>
                        <option value="Préstamo">Préstamo</option>
                        <option value="Otro">Otro</option>
                    </select>
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Cliente asociado (ID)*</label>
                    <input type="text" class="product-client-id-input" l
                    <datalist id="clientsIdDatalist"></datalist>
                    <div class="error-message"></div>
                </div>

                <div class="form-group">
                    <label>Monto desembolsado*</label>
                    <input type="number" min="0" step="0.01" class="prod
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Monto investigado</label>
                    <input type="number" min="0" step="0.01" class="prod
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Monto pérdida estimada</label>
                    <input type="number" min="0" step="0.01" class="prod
                    <div class="error-message"></div>
                </div>

                <div class="form-group">
                    <label>Fecha de ocurrencia*</label>
                    <input type="date" class="product-occurrence-date-in
                    <div class="error-message date-tooltip">
                        Ingresar ambas fechas para validar la secuencia.
                    </div>
                </div>
                <div class="form-group">
                    <label>Fecha de descubrimiento*</label>
                    <input type="date" class="product-discovery-date-inp
                    <div class="error-message"></div>
                </div>
                <div class="form-group">
                    <label>Flag (opcional)</label>
                    <input type="text" class="product-flag-input"
                           placeholder="Uso interno, opcional">
                </div>

                <div class="form-group">
                    <label>Accionado (opcional)</label>
```

```
                    <input type="text" class="product-accionado-input"
                           placeholder="Área / persona accionada">
                </div>
                <div class="form-group full-width">
                    <label>Notas</label>
                    <input type="text" class="product-notes-input">
                </div>
            </div>
        </div>
    </template>

</main>

<script src="script.js"></script>
</body>
</html>
```

---

style.css

css Copy code

```css
/* Layout básico */
*,
*::before,
*::after {
    box-sizing: border-box;
}

body {
    margin: 0;
    padding: 0;
    font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI
    background-color: #f5f5f7;
    color: #222;
}

/* Scroll vertical siempre permitido (evita cortes raros) */
html, body {
    height: 100%;
```

```
}

.app-header {
    padding: 1rem 1.5rem;
    background: #111827;
    color: #f9fafb;
    border-bottom: 3px solid #1d4ed8;
}

.app-header h1 {
    margin: 0;
    font-size: 1.6rem;
}

.app-header .subtitle {
    margin: 0.25rem 0 0;
    font-size: 0.9rem;
    opacity: 0.8;
}

.app-main {
    max-width: 1200px;
    margin: 1rem auto 2rem;
    padding: 0 1rem;
}

/* Tarjetas */
.card {
    background: #ffffff;
    border-radius: 0.75rem;
    padding: 1rem 1.25rem;
    margin-bottom: 1rem;
    box-shadow: 0 1px 4px rgba(15, 23, 42, 0.08);
    border: 1px solid #e5e7eb;
}

/* Títulos de sección */
.card h2 {
    margin-top: 0;
    margin-bottom: 0.75rem;
    font-size: 1.2rem;
}

/* Grids */
.grid-2 {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 0.75rem 1rem;
}

.grid-3 {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 0.75rem 1rem;
}

/* Formulario */
.form-group {
    display: flex;
    flex-direction: column;
    gap: 0.25rem;
}

.form-group.full-width {
    grid-column: 1 / -1;
}

label {
    font-size: 0.85rem;
    font-weight: 600;
    color: #374151;
}

input[type="text"],
input[type="date"],
input[type="number"],
select,
textarea {
    padding: 0.4rem 0.5rem;
    border-radius: 0.4rem;
```

```css
    border: 1px solid #d1d5db;
    font-size: 0.9rem;
    outline: none;
    background-color: #f9fafb;
}

input:focus,
select:focus,
textarea:focus {
    border-color: #2563eb;
    box-shadow: 0 0 0 1px rgba(37, 99, 235, 0.25);
    background-color: #ffffff;
}

textarea {
    resize: vertical;
}

/* Botones (sin animación de “achicado”) */
.btn,
.icon-btn {
    border-radius: 999px;
    border: none;
    cursor: pointer;
    font-size: 0.9rem;
    display: inline-flex;
    align-items: center;
    justify-content: center;
    gap: 0.35rem;
    padding: 0.4rem 0.9rem;
    transition: background-color 0.15s ease, box-shadow 0.15s ease, colo
    user-select: none;
}

.btn {
    background-color: #e5e7eb;
    color: #111827;
}

.btn.primary {
    background-color: #2563eb;
    color: #f9fafb;
}

.btn.secondary {
    background-color: #e5e7eb;
    color: #111827;
}

.btn.danger {
    background-color: #b91c1c;
    color: #f9fafb;
}

.btn:hover,
.icon-btn:hover {
    /* SOLO color / sombra: NO transform, NO scale */
    box-shadow: 0 2px 6px rgba(15, 23, 42, 0.15);
    filter: brightness(0.97);
}

.btn:active,
.icon-btn:active {
    box-shadow: 0 1px 3px rgba(15, 23, 42, 0.18) inset;
}

/* Icon button */
.icon-btn {
    background-color: #f3f4f6;
    color: #4b5563;
    padding: 0.25rem 0.5rem;
    border-radius: 999px;
}

.icon-btn:hover {
    background-color: #e5e7eb;
}

/* File input “bonito” */
.file-label {
```

```css
    position: relative;
    overflow: hidden;
    display: inline-flex;
    align-items: center;
    justify-content: center;
    padding: 0.35rem 0.8rem;
    border-radius: 999px;
    background-color: #f3f4f6;
    border: 1px dashed #d1d5db;
    font-size: 0.8rem;
    cursor: pointer;
}

.file-label input[type="file"] {
    position: absolute;
    inset: 0;
    opacity: 0;
    cursor: pointer;
}

/* Cabecera de secciones */
.section-header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 0.5rem;
    margin-bottom: 0.5rem;
}

.section-actions {
    display: flex;
    align-items: center;
    gap: 0.5rem;
}

/* Contenedor de filas dinámicas */
.instance-container {
    display: flex;
    flex-direction: column;
    gap: 0.75rem;
    margin-top: 0.5rem;
}

/* Fila individual (cliente, colaborador, producto) */
.instance-row {
    border-radius: 0.6rem;
    border: 1px solid #e5e7eb;
    padding: 0.7rem 0.8rem;
    background-color: #f9fafb;
}

.row-header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    margin-bottom: 0.35rem;
}

.row-header h3 {
    margin: 0;
    font-size: 1rem;
}

/* Ayudas y mensajes */
.helper-text {
    font-size: 0.8rem;
    color: #6b7280;
    margin-top: 0.25rem;
}

/* Errores */
.error-message {
    font-size: 0.75rem;
    color: #b91c1c;
    min-height: 0.8rem;
}

input.error,
select.error,
textarea.error {
    border-color: #b91c1c;
    box-shadow: 0 0 0 1px rgba(185, 28, 28, 0.18);
    background-color: #fef2f2;
}

/* Tooltip de fechas: texto fijo, no flotante (sin hover raro) */
.date-tooltip {
    font-style: italic;
}

/* Resumen */
.summary-wrapper {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
    gap: 0.75rem;
}

.summary-block h3 {
    margin-top: 0;
    font-size: 1rem;
}

/* Tablas */
table {
    width: 100%;
    border-collapse: collapse;
    font-size: 0.8rem;
    background-color: #fff;
}

th,
td {
    border: 1px solid #e5e7eb;
    padding: 0.3rem 0.35rem;
    text-align: left;
}

th {
    background-color: #f3f4f6;
    font-weight: 600;
```

```css
}

/* Controles y cajas globales */
.controls-grid {
    display: grid;
    grid-template-columns: minmax(0, 2fr) minmax(0, 2fr) minmax(0, 2fr);
    gap: 1rem;
    align-items: flex-start;
}

.button-group-vertical {
    display: flex;
    flex-direction: column;
    gap: 0.4rem;
}

.global-error,
.global-info {
    margin-top: 0.75rem;
    padding: 0.5rem 0.75rem;
    border-radius: 0.5rem;
    font-size: 0.85rem;
}

.global-error {
    background-color: #fef2f2;
    border: 1px solid #fecaca;
    color: #991b1b;
}

.global-info {
    background-color: #ecfdf3;
    border: 1px solid #bbf7d0;
    color: #166534;
}

/* Responsivo */
@media (max-width: 768px) {
    .controls-grid {
        grid-template-columns: 1fr;
    }

    .section-header {
        flex-direction: column;
        align-items: flex-start;
    }
}
```

---

script.js

javascript

Copy code

```javascript
// script.js
// Gestor de Casos de Fraude — App web simple (HTML/CSS/JS puro)

/**
 * Estructura en memoria del caso.
 * Se sincroniza con el formulario y con localStorage.
 */
let caseState = {
    caseInfo: {
        caseId: "",
        caseTitle: "",
        leadInvestigator: "",
        caseStartDate: "",
        caseSummary: ""
    },
    clients: [],
    teamMembers: [],
    products: [],
    checkpoints: [] // snapshots de estado completo
};

const STORAGE_KEY = "fraud_case_current_data_v1";
const CHECKPOINTS_KEY = "fraud_case_checkpoints_v1";

document.addEventListener("DOMContentLoaded", () => {
    // Referencias generales
    const clientsContainer = document.getElementById("clientsContainer")
    const teamContainer = document.getElementById("teamContainer");
    const productsContainer = document.getElementById("productsContainer

    const addClientBtn = document.getElementById("addClientBtn");
    const addTeamMemberBtn = document.getElementById("addTeamMemberBtn")
    const addProductBtn = document.getElementById("addProductBtn");

    const clientsCsvInput = document.getElementById("clientsCsvInput");
    const teamCsvInput = document.getElementById("teamCsvInput");
    const productsCsvInput = document.getElementById("productsCsvInput")

    const generateReportBtn = document.getElementById("generateReportBtn
    const saveCheckpointBtn = document.getElementById("saveCheckpointBtn
    const restoreCheckpointBtn = document.getElementById("restoreCheckpo
    const checkpointSelect = document.getElementById("checkpointSelect")
    const clearAllBtn = document.getElementById("clearAllBtn");

    const globalErrorBox = document.getElementById("globalErrorBox");
    const globalInfoBox = document.getElementById("globalInfoBox");

    // Plantillas
    const clientRowTemplate = document.getElementById("clientRowTemplate
    const teamRowTemplate = document.getElementById("teamRowTemplate");
    const productRowTemplate = document.getElementById("productRowTempla

    // === Funciones utilitarias ===

    function showGlobalError(msg) {
        if (!msg) {
            globalErrorBox.hidden = true;
            globalErrorBox.textContent = "";
            return;
        }
        globalErrorBox.hidden = false;
        globalErrorBox.textContent = msg;
    }

    function showGlobalInfo(msg) {
        if (!msg) {
            globalInfoBox.hidden = true;
            globalInfoBox.textContent = "";
            return;
        }
        globalInfoBox.hidden = false;
        globalInfoBox.textContent = msg;
        // Se oculta solo el info después de unos segundos
        setTimeout(() => {
            globalInfoBox.hidden = true;
        }, 4000);
    }

    function clearGlobalMessages() {
        showGlobalError("");
        showGlobalInfo("");
    }

    function deepClone(obj) {
        return JSON.parse(JSON.stringify(obj));
    }

    // === Auto-save con localStorage ===

    function saveToLocalStorage() {
        try {
            const toStore = {
                caseInfo: caseState.caseInfo,
                clients: caseState.clients,
                teamMembers: caseState.teamMembers,
                products: caseState.products
            };
            localStorage.setItem(STORAGE_KEY, JSON.stringify(toStore));
        } catch (e) {
            console.warn("No se pudo guardar en localStorage:", e);
        }
    }

    function loadFromLocalStorage() {
        try {
            const raw = localStorage.getItem(STORAGE_KEY);
            if (!raw) return;
            const parsed = JSON.parse(raw);
            caseState.caseInfo = parsed.caseInfo || caseState.caseInfo;
            caseState.clients = parsed.clients || [];
            caseState.teamMembers = parsed.teamMembers || [];
            caseState.products = parsed.products || [];
        } catch (e) {
            console.warn("No se pudo leer de localStorage:", e);
        }
    }

    function saveCheckpointsToLocalStorage() {
        try {
            localStorage.setItem(CHECKPOINTS_KEY, JSON.stringify(caseSta
        } catch (e) {
            console.warn("No se pudieron guardar checkpoints:", e);
        }
    }

    function loadCheckpointsFromLocalStorage() {
        try {
            const raw = localStorage.getItem(CHECKPOINTS_KEY);
            if (!raw) return;
            const parsed = JSON.parse(raw);
            caseState.checkpoints = Array.isArray(parsed) ? parsed : [];
        } catch (e) {
            console.warn("No se pudieron leer checkpoints:", e);
        }
    }

    // === Refrescado de datalist de clientes (para asociar productos) =

    function refreshClientsDatalist() {
        const datalist = document.getElementById("clientsIdDatalist");
        datalist.innerHTML = "";
        caseState.clients.forEach(c => {
            const opt = document.createElement("option");
            opt.value = c.id;
            opt.label = c.name;
            datalist.appendChild(opt);
        });
```

```
}

// === Actualizar índices visibles (Cliente 1, Cliente 2, etc.) ===

function refreshRowIndices(containerSelector) {
    const container = document.querySelector(containerSelector);
    if (!container) return;
    const rows = container.querySelectorAll(".instance-row");
    rows.forEach((row, idx) => {
        const span = row.querySelector(".row-index");
        if (span) span.textContent = String(idx + 1);
    });
}

// === Render de CLIENTES desde estado ===

function renderClients() {
    clientsContainer.innerHTML = "";
    caseState.clients.forEach(c => {
        const clone = clientRowTemplate.content.cloneNode(true);
        const row = clone.querySelector(".client-row");

        row.dataset.clientId = c.id;

        const idInput = row.querySelector(".client-id-input");
        const nameInput = row.querySelector(".client-name-input");
        const docTypeInput = row.querySelector(".client-doc-type-inp
        const docNumberInput = row.querySelector(".client-doc-number
        const notesInput = row.querySelector(".client-notes-input");

        idInput.value = c.id;
        nameInput.value = c.name;
        docTypeInput.value = c.docType;
        docNumberInput.value = c.docNumber;
        notesInput.value = c.notes || "";

        attachClientRowListeners(row);
        clientsContainer.appendChild(clone);
    });
    refreshRowIndices("#clientsContainer");
```

```
}

function attachClientRowListeners(row) {
    const removeBtn = row.querySelector(".remove-btn");
    removeBtn.addEventListener("click", () => {
        const idInput = row.querySelector(".client-id-input");
        const id = idInput.value.trim();
        caseState.clients = caseState.clients.filter(c => c.id !== i
        saveToLocalStorage();
        renderClients();
        refreshSummaryTables();
        refreshClientsDatalist();
    });

    const inputs = row.querySelectorAll("input, select");
    inputs.forEach(input => {
        input.addEventListener("input", () => {
            updateClientsFromDOM();
            refreshSummaryTables();
            refreshClientsDatalist();
            saveToLocalStorage();
        });
    });
}

function updateClientsFromDOM() {
    const rows = clientsContainer.querySelectorAll(".client-row");
    const newClients = [];
    rows.forEach(row => {
        const id = row.querySelector(".client-id-input").value.trim(
        const name = row.querySelector(".client-name-input").value.t
        const docType = row.querySelector(".client-doc-type-input").
        const docNumber = row.querySelector(".client-doc-number-inpu
        const notes = row.querySelector(".client-notes-input").value
        newClients.push({ id, name, docType, docNumber, notes });
    });
    caseState.clients = newClients;
}

function addClient(initialData = null) {
    const client = initialData || {
        id: "",
        name: "",
        docType: "",
        docNumber: "",
        notes: ""
    };
    caseState.clients.push(client);
    renderClients();
    refreshSummaryTables();
    refreshClientsDatalist();
    saveToLocalStorage();
}

// === Render de COLABORADORES desde estado ===

function renderTeamMembers() {
    teamContainer.innerHTML = "";
    caseState.teamMembers.forEach(t => {
        const clone = teamRowTemplate.content.cloneNode(true);
        const row = clone.querySelector(".team-row");
        row.dataset.teamId = t.id;

        const idInput = row.querySelector(".team-id-input");
        const nameInput = row.querySelector(".team-name-input");
        const roleInput = row.querySelector(".team-role-input");

        idInput.value = t.id;
        nameInput.value = t.name;
        roleInput.value = t.role || "";

        attachTeamRowListeners(row);
        teamContainer.appendChild(clone);
    });
    refreshRowIndices("#teamContainer");
}

function attachTeamRowListeners(row) {
    const removeBtn = row.querySelector(".remove-btn");
    removeBtn.addEventListener("click", () => {
        const idInput = row.querySelector(".team-id-input");
        const id = idInput.value.trim();
        caseState.teamMembers = caseState.teamMembers.filter(t => t.
        saveToLocalStorage();
        renderTeamMembers();
        refreshSummaryTables();
    });

    const inputs = row.querySelectorAll("input");
    inputs.forEach(input => {
        input.addEventListener("input", () => {
            updateTeamMembersFromDOM();
            refreshSummaryTables();
            saveToLocalStorage();
        });
    });
}

function updateTeamMembersFromDOM() {
    const rows = teamContainer.querySelectorAll(".team-row");
    const newTeam = [];
    rows.forEach(row => {
        const id = row.querySelector(".team-id-input").value.trim();
        const name = row.querySelector(".team-name-input").value.tri
        const role = row.querySelector(".team-role-input").value.tri
        newTeam.push({ id, name, role });
    });
    caseState.teamMembers = newTeam;
}

function addTeamMember(initialData = null) {
    const tm = initialData || {
        id: "",
        name: "",
        role: ""
    };
    caseState.teamMembers.push(tm);
    renderTeamMembers();
    refreshSummaryTables();
    saveToLocalStorage();
```

```
}

// === Render de PRODUCTOS desde estado ===

function renderProducts() {
    productsContainer.innerHTML = "";
    caseState.products.forEach(p => {
        const clone = productRowTemplate.content.cloneNode(true);
        const row = clone.querySelector(".product-row");
        row.dataset.productId = p.id;

        row.querySelector(".product-id-input").value = p.id;
        row.querySelector(".product-type-input").value = p.type;
        row.querySelector(".product-client-id-input").value = p.clie
        row.querySelector(".product-amount-input").value = p.amount;
        row.querySelector(".product-investigation-amount-input").val
        row.querySelector(".product-loss-amount-input").value = p.lo
        row.querySelector(".product-occurrence-date-input").value =
        row.querySelector(".product-discovery-date-input").value = p
        row.querySelector(".product-flag-input").value = p.flag || "
        row.querySelector(".product-accionado-input").value = p.acci
        row.querySelector(".product-notes-input").value = p.notes ||

        attachProductRowListeners(row);
        productsContainer.appendChild(clone);
    });
    refreshRowIndices("#productsContainer");
}

function attachProductRowListeners(row) {
    const removeBtn = row.querySelector(".remove-btn");
    removeBtn.addEventListener("click", () => {
        const id = row.querySelector(".product-id-input").value.trim
        caseState.products = caseState.products.filter(p => p.id !==
        saveToLocalStorage();
        renderProducts();
        refreshSummaryTables();
    });

    const inputs = row.querySelectorAll("input, select");
    inputs.forEach(input => {
        // Importante: NO cambiamos display ni tamaño del input en e
        input.addEventListener("input", () => {
            updateProductsFromDOM();
            refreshSummaryTables();
            saveToLocalStorage();
        });
        input.addEventListener("change", () => {
            updateProductsFromDOM();
            refreshSummaryTables();
            saveToLocalStorage();
        });
    });
}

function updateProductsFromDOM() {
    const rows = productsContainer.querySelectorAll(".product-row");
    const newProducts = [];
    rows.forEach(row => {
        const id = row.querySelector(".product-id-input").value.trim
        const type = row.querySelector(".product-type-input").value.
        const clientId = row.querySelector(".product-client-id-input
        const amount = parseFloat(row.querySelector(".product-amount
        const investigationAmount = parseFloat(row.querySelector(".p
        const lossAmount = parseFloat(row.querySelector(".product-lo
        const occurrenceDate = row.querySelector(".product-occurrenc
        const discoveryDate = row.querySelector(".product-discovery-
        const flag = row.querySelector(".product-flag-input").value.
        const accionado = row.querySelector(".product-accionado-inpu
        const notes = row.querySelector(".product-notes-input").valu

        newProducts.push({
            id,
            type,
            clientId,
            amount,
            investigationAmount,
            lossAmount,
            occurrenceDate,
            discoveryDate,
            flag,
            accionado,
            notes
        });
    });
    caseState.products = newProducts;
}

function addProduct(initialData = null) {
    const p = initialData || {
        id: "",
        type: "",
        clientId: "",
        amount: 0,
        investigationAmount: 0,
        lossAmount: 0,
        occurrenceDate: "",
        discoveryDate: "",
        flag: "",
        accionado: "",
        notes: ""
    };
    caseState.products.push(p);
    renderProducts();
    refreshSummaryTables();
    saveToLocalStorage();
}

// === Info del caso desde/ hacia DOM ===

function readCaseInfoFromDOM() {
    caseState.caseInfo.caseId = document.getElementById("caseId").va
    caseState.caseInfo.caseTitle = document.getElementById("caseTitl
    caseState.caseInfo.leadInvestigator = document.getElementById("l
    caseState.caseInfo.caseStartDate = document.getElementById("case
    caseState.caseInfo.caseSummary = document.getElementById("caseSu
}

function renderCaseInfo() {
    document.getElementById("caseId").value = caseState.caseInfo.cas
    document.getElementById("caseTitle").value = caseState.caseInfo.
    document.getElementById("leadInvestigator").value = caseState.ca
    document.getElementById("caseStartDate").value = caseState.caseI
    document.getElementById("caseSummary").value = caseState.caseInf
}

function attachCaseInfoListeners() {
    ["caseId", "caseTitle", "leadInvestigator", "caseStartDate", "ca
        document.getElementById(id).addEventListener("input", () =>
            readCaseInfoFromDOM();
            saveToLocalStorage();
        });
    });
}

// === Resumen / Tablas compactas ===

function refreshSummaryTables() {
    // Clientes
    const clientsBody = document.querySelector("#summaryClientsTable
    clientsBody.innerHTML = "";
    caseState.clients.forEach(c => {
        const tr = document.createElement("tr");
        tr.innerHTML = `
            <td>${c.id || ""}</td>
            <td>${c.name || ""}</td>
            <td>${c.docType || ""}</td>
            <td>${c.docNumber || ""}</td>
        `;
        clientsBody.appendChild(tr);
    });

    // Colaboradores
    const teamBody = document.querySelector("#summaryTeamTable tbody
    teamBody.innerHTML = "";
    caseState.teamMembers.forEach(t => {
        const tr = document.createElement("tr");
        tr.innerHTML = `
            <td>${t.id || ""}</td>
            <td>${t.name || ""}</td>
            <td>${t.role || ""}</td>
        `;
        teamBody.appendChild(tr);
    });

    // Productos
    const productsBody = document.querySelector("#summaryProductsTab
    productsBody.innerHTML = "";
    caseState.products.forEach(p => {
        const tr = document.createElement("tr");
        tr.innerHTML = `
            <td>${p.id || ""}</td>
            <td>${p.type || ""}</td>
            <td>${p.clientId || ""}</td>
            <td>${p.amount != null ? p.amount : ""}</td>
            <td>${p.investigationAmount != null ? p.investigationAmo
            <td>${p.lossAmount != null ? p.lossAmount : ""}</td>
        `;
        productsBody.appendChild(tr);
    });
}

// === Validaciones ===

function clearFieldError(input) {
    input.classList.remove("error");
    const fg = input.closest(".form-group");
    if (!fg) return;
    const err = fg.querySelector(".error-message");
    if (err) err.textContent = "";
}

function setFieldError(input, msg) {
    input.classList.add("error");
    const fg = input.closest(".form-group");
    if (!fg) return;
    const err = fg.querySelector(".error-message");
    if (err) err.textContent = msg || "";
}
```

```
        function validateAll() {
            clearGlobalMessages();
            let valid = true;

            // Limpia errores previos
            document.querySelectorAll(".error").forEach(el => el.classList.r
            document.querySelectorAll(".error-message").forEach(el => {
                if (!el.classList.contains("date-tooltip")) el.textContent =
            });

            // Validar clientes: ID único, campos obligatorios
            const clientIds = new Set();
            document.querySelectorAll(".client-row").forEach(row => {
                const idInput = row.querySelector(".client-id-input");
                const nameInput = row.querySelector(".client-name-input");
                const docTypeInput = row.querySelector(".client-doc-type-inp
                const docNumberInput = row.querySelector(".client-doc-number

                if (!idInput.value.trim()) {
                    setFieldError(idInput, "ID obligatorio.");
                    valid = false;
                } else if (clientIds.has(idInput.value.trim())) {
                    setFieldError(idInput, "ID duplicado.");
                    valid = false;
                } else {
                    clientIds.add(idInput.value.trim());
                }

                if (!nameInput.value.trim()) {
                    setFieldError(nameInput, "Nombre obligatorio.");
                    valid = false;
                }
                if (!docTypeInput.value.trim()) {
                    setFieldError(docTypeInput, "Tipo de documento obligator
                    valid = false;
                }
                if (!docNumberInput.value.trim()) {
                    setFieldError(docNumberInput, "Número de documento oblig
                    valid = false;
                }
            });

            // Validar colaboradores: ID único, nombre obligatorio
            const teamIds = new Set();
            document.querySelectorAll(".team-row").forEach(row => {
                const idInput = row.querySelector(".team-id-input");
                const nameInput = row.querySelector(".team-name-input");

                if (!idInput.value.trim()) {
                    setFieldError(idInput, "ID obligatorio.");
                    valid = false;
                } else if (teamIds.has(idInput.value.trim())) {
                    setFieldError(idInput, "ID duplicado.");
                    valid = false;
                } else {
                    teamIds.add(idInput.value.trim());
                }

                if (!nameInput.value.trim()) {
                    setFieldError(nameInput, "Nombre obligatorio.");
                    valid = false;
                }
            });

            // Validar productos
            const productIds = new Set();
            document.querySelectorAll(".product-row").forEach(row => {
                const idInput = row.querySelector(".product-id-input");
                const typeInput = row.querySelector(".product-type-input");
                const clientIdInput = row.querySelector(".product-client-id-
                const amountInput = row.querySelector(".product-amount-input
                const invAmountInput = row.querySelector(".product-investiga
                const lossAmountInput = row.querySelector(".product-loss-amo
                const occDateInput = row.querySelector(".product-occurrence-
                const discDateInput = row.querySelector(".product-discovery-

                if (!idInput.value.trim()) {
                    setFieldError(idInput, "ID obligatorio.");
                    valid = false;
                } else if (productIds.has(idInput.value.trim())) {
```

```
    setFieldError(idInput, "ID duplicado.");
    valid = false;
} else {
    productIds.add(idInput.value.trim());
}

if (!typeInput.value.trim()) {
    setFieldError(typeInput, "Tipo obligatorio.");
    valid = false;
}

if (!clientIdInput.value.trim()) {
    setFieldError(clientIdInput, "Cliente asociado obligator
    valid = false;
} else if (!clientIds.has(clientIdInput.value.trim())) {
    setFieldError(clientIdInput, "Cliente no coincide con ni
    valid = false;
}

const amt = parseFloat(amountInput.value);
if (isNaN(amt) || amt < 0) {
    setFieldError(amountInput, "Monto inválido.");
    valid = false;
}

const inv = invAmountInput.value.trim() ? parseFloat(invAmou
if (invAmountInput.value.trim() && (isNaN(inv) || inv < 0))
    setFieldError(invAmountInput, "Monto inválido.");
    valid = false;
}

const loss = lossAmountInput.value.trim() ? parseFloat(lossA
if (lossAmountInput.value.trim() && (isNaN(loss) || loss < 0
    setFieldError(lossAmountInput, "Monto inválido.");
    valid = false;
}

// Fechas: ambas obligatorias, y descubrimiento >= ocurrenci
const occVal = occDateInput.value;
const discVal = discDateInput.value;
```

```
        if (!occVal) {
            setFieldError(occDateInput, "Fecha de ocurrencia obligat
            valid = false;
        }

        if (!discVal) {
            setFieldError(discDateInput, "Fecha de descubrimiento ob
            valid = false;
        }

        if (occVal && discVal) {
            const occDate = new Date(occVal);
            const discDate = new Date(discVal);
            if (discDate < occDate) {
                setFieldError(discDateInput, "Descubrimiento no pued
                valid = false;
            }
        }
    });

    if (!valid) {
        showGlobalError("Hay errores de validación. Por favor, corre
    } else {
        showGlobalInfo("Validación correcta.");
    }

    return valid;
}

// === CSV parsing ===

function parseCsv(text) {
    // parser simple: separa por líneas y luego por coma.
    const lines = text.split(/\r?\n/).map(l => l.trim()).filter(l =>
    if (lines.length === 0) return [];
    const rows = [];
    for (let i = 0; i < lines.length; i++) {
        const line = lines[i];
        // manejo muy simple, sin comillas escapadas
        const cells = line.split(",").map(c => c.trim());
        rows.push(cells);
    }
    return rows;
}

function handleClientsCsv(text) {
    const rows = parseCsv(text);
    // Suponemos columnas: id, nombre, tipo_doc, num_doc, notas
    rows.forEach(cells => {
        if (cells.length < 4) return; // fila inválida
        const [id, name, docType, docNumber, notes] = cells;
        if (!id) return;
        // evitar duplicados
        if (caseState.clients.some(c => c.id === id)) {
            // skip duplicado; en versión futura se podría mergear
            return;
        }
        caseState.clients.push({
            id: id.trim(),
            name: (name || "").trim(),
            docType: (docType || "").trim(),
            docNumber: (docNumber || "").trim(),
            notes: (notes || "").trim()
        });
    });
    renderClients();
    refreshSummaryTables();
    refreshClientsDatalist();
    saveToLocalStorage();
    showGlobalInfo("Carga masiva de clientes completada.");
}

function handleTeamCsv(text) {
    const rows = parseCsv(text);
    // Columnas: id, nombre, rol
    rows.forEach(cells => {
        if (cells.length < 2) return;
        const [id, name, role] = cells;
        if (!id) return;
        if (caseState.teamMembers.some(t => t.id === id)) {
            return;
        }
        caseState.teamMembers.push({
            id: id.trim(),
            name: (name || "").trim(),
            role: (role || "").trim()
        });
    });
    renderTeamMembers();
    refreshSummaryTables();
    saveToLocalStorage();
    showGlobalInfo("Carga masiva de colaboradores completada.");
}

function handleProductsCsv(text) {
    const rows = parseCsv(text);
    // Columnas esperadas (mínimas):
    // id, tipo, clientId, amount, occDate, discDate, invAmount, los
    rows.forEach(cells => {
        if (cells.length < 5) return;
        const [
            id,
            type,
            clientId,
            amountStr,
            occDate,
            discDate,
            invAmountStr,
            lossAmountStr,
            flag,
            accionado,
            notes
        ] = cells;

        if (!id) return;
        if (caseState.products.some(p => p.id === id)) {
            return;
        }
```

```
            const amount = parseFloat(amountStr) || 0;
            const invAmount = invAmountStr ? (parseFloat(invAmountStr) |
            const lossAmount = lossAmountStr ? (parseFloat(lossAmountStr

            caseState.products.push({
                id: id.trim(),
                type: (type || "").trim(),
                clientId: (clientId || "").trim(),
                amount,
                investigationAmount: invAmount,
                lossAmount,
                occurrenceDate: (occDate || "").trim(),
                discoveryDate: (discDate || "").trim(),
                flag: (flag || "").trim(),
                accionado: (accionado || "").trim(),
                notes: (notes || "").trim()
            });
        });
        renderProducts();
        refreshSummaryTables();
        saveToLocalStorage();
        showGlobalInfo("Carga masiva de productos completada.");
    }

    function readFileAsText(file, callback) {
        const reader = new FileReader();
        reader.onload = e => callback(e.target.result);
        reader.onerror = () => showGlobalError("Error leyendo el archivo
        reader.readAsText(file);
    }

    // === Checkpoints ===

    function refreshCheckpointSelect() {
        checkpointSelect.innerHTML = `<option value="">(Ningún checkpoin
        caseState.checkpoints.forEach((cp, idx) => {
            const opt = document.createElement("option");
            const ts = cp.timestamp || new Date().toISOString();
            opt.value = String(idx);
            opt.textContent = `${idx + 1} — ${ts}`;
            checkpointSelect.appendChild(opt);
        });
    }

    function saveCheckpoint() {
        readCaseInfoFromDOM();
        updateClientsFromDOM();
        updateTeamMembersFromDOM();
        updateProductsFromDOM();

        const snapshot = {
            data: deepClone({
                caseInfo: caseState.caseInfo,
                clients: caseState.clients,
                teamMembers: caseState.teamMembers,
                products: caseState.products
            }),
            timestamp: new Date().toLocaleString()
        };
        caseState.checkpoints.push(snapshot);
        saveCheckpointsToLocalStorage();
        refreshCheckpointSelect();
        showGlobalInfo("Checkpoint guardado.");
    }

    function restoreSelectedCheckpoint() {
        const idxStr = checkpointSelect.value;
        if (!idxStr) {
            showGlobalError("Selecciona un checkpoint para restaurar.");
            return;
        }
        const idx = parseInt(idxStr, 10);
        if (isNaN(idx) || !caseState.checkpoints[idx]) {
            showGlobalError("Checkpoint inválido.");
            return;
        }
        const snapshot = caseState.checkpoints[idx].data;
        caseState.caseInfo = deepClone(snapshot.caseInfo);
        caseState.clients = deepClone(snapshot.clients);
        caseState.teamMembers = deepClone(snapshot.teamMembers);
        caseState.products = deepClone(snapshot.products);

        renderCaseInfo();
        renderClients();
        renderTeamMembers();
        renderProducts();
        refreshSummaryTables();
        refreshClientsDatalist();
        saveToLocalStorage();
        showGlobalInfo("Checkpoint restaurado.");
    }

    // === Limpieza total ===

    function clearAll() {
        if (!confirm("¿Seguro que quieres limpiar todo el formulario act
            return;
        }
        caseState.caseInfo = {
            caseId: "",
            caseTitle: "",
            leadInvestigator: "",
            caseStartDate: "",
            caseSummary: ""
        };
        caseState.clients = [];
        caseState.teamMembers = [];
        caseState.products = [];

        renderCaseInfo();
        renderClients();
        renderTeamMembers();
        renderProducts();
        refreshSummaryTables();
        refreshClientsDatalist();
        saveToLocalStorage();
        showGlobalInfo("Formulario vaciado.");
    }

    // === Reporte (.doc) ===
    function generateReport() {
        if (!validateAll()) {
            return;
        }

        // Aseguramos estado actualizado
        readCaseInfoFromDOM();
        updateClientsFromDOM();
        updateTeamMembersFromDOM();
        updateProductsFromDOM();

        const ci = caseState.caseInfo;
        const clients = caseState.clients;
        const team = caseState.teamMembers;
        const products = caseState.products;

        // Construimos HTML sencillo que Word puede interpretar
        let html = `
<html>
<head>
<meta charset="UTF-8">
<title>Reporte de caso ${ci.caseId || ""}</title>
<style>
body { font-family: Arial, sans-serif; font-size: 11pt; }
h1, h2, h3 { color: #111; }
table { border-collapse: collapse; width: 100%; margin-bottom: 12px; }
th, td { border: 1px solid #999; padding: 4px 6px; font-size: 10pt; }
th { background: #eee; }
</style>
</head>
<body>
<h1>Reporte de Caso de Fraude</h1>
<h2>1. Información general</h2>
<p><strong>ID del caso:</strong> ${ci.caseId || ""}</p>
<p><strong>Título:</strong> ${ci.caseTitle || ""}</p>
<p><strong>Investigador responsable:</strong> ${ci.leadInvestigator || "
<p><strong>Fecha de inicio:</strong> ${ci.caseStartDate || ""}</p>
<p><strong>Resumen:</strong><br>${(ci.caseSummary || "").replace(/\n/g,
<h2>2. Clientes</h2>
<table>
<thead>
<tr>
<th>ID</th><th>Nombre</th><th>Tipo doc.</th><th>N° documento</th><th>Not
</tr>
</thead>
<tbody>`;

        clients.forEach(c => {
            html += `
<tr>
<td>${c.id || ""}</td>
<td>${c.name || ""}</td>
<td>${c.docType || ""}</td>
<td>${c.docNumber || ""}</td>
<td>${c.notes || ""}</td>
</tr>`;
        });

        html += `
</tbody></table>

<h2>3. Colaboradores / Investigadores</h2>
<table>
<thead>
<tr>
<th>ID</th><th>Nombre</th><th>Rol</th>
</tr>
</thead>
<tbody>`;

        team.forEach(t => {
            html += `
<tr>
<td>${t.id || ""}</td>
<td>${t.name || ""}</td>
<td>${t.role || ""}</td>
</tr>`;
        });
        html += `
</tbody></table>

<h2>4. Productos</h2>
<table>
<thead>
<tr>
<th>ID producto</th>
<th>Tipo</th>
<th>Cliente asociado</th>
<th>Monto desembolsado</th>
<th>Monto investigado</th>
<th>Monto pérdida</th>
<th>Fecha ocurrencia</th>
<th>Fecha descubrimiento</th>
<th>Flag</th>
<th>Accionado</th>
<th>Notas</th>
</tr>
</thead>
<tbody>`;

        products.forEach(p => {
            html += `
<tr>
<td>${p.id || ""}</td>
<td>${p.type || ""}</td>
<td>${p.clientId || ""}</td>
<td>${p.amount != null ? p.amount : ""}</td>
<td>${p.investigationAmount != null ? p.investigationAmount : ""}</td>
<td>${p.lossAmount != null ? p.lossAmount : ""}</td>
<td>${p.occurrenceDate || ""}</td>
<td>${p.discoveryDate || ""}</td>
<td>${p.flag || ""}</td>
<td>${p.accionado || ""}</td>
<td>${p.notes || ""}</td>
</tr>`;
        });
        html += `
</tbody></table>

<p><em>Este reporte fue generado automáticamente por la herramienta de g
</body>
</html>`;

        const blob = new Blob([html], { type: "application/msword" });
        const url = URL.createObjectURL(blob);

        const a = document.createElement("a");
        const caseId = ci.caseId || "SIN_ID";
        a.href = url;
        a.download = `CaseReport_${caseId}.doc`;
        document.body.appendChild(a);
        a.click();
        document.body.removeChild(a);
        URL.revokeObjectURL(url);

        showGlobalInfo("Reporte generado.");
    }

    // === Inicialización ===

    loadFromLocalStorage();
    loadCheckpointsFromLocalStorage();
    renderCaseInfo();
    renderClients();
    renderTeamMembers();
    renderProducts();
    refreshSummaryTables();
    refreshClientsDatalist();
    refreshCheckpointSelect();
    attachCaseInfoListeners();

    // Si no hay al menos una fila, creamos una por defecto
    if (caseState.clients.length === 0) addClient();
    if (caseState.teamMembers.length === 0) addTeamMember();
    if (caseState.products.length === 0) addProduct();
```

```
// === Event listeners ===

addClientBtn.addEventListener("click", () => {
    clearGlobalMessages();
    addClient();
});

addTeamMemberBtn.addEventListener("click", () => {
    clearGlobalMessages();
    addTeamMember();
});

addProductBtn.addEventListener("click", () => {
    clearGlobalMessages();
    addProduct();
});

clientsCsvInput.addEventListener("change", () => {
    clearGlobalMessages();
    const file = clientsCsvInput.files[0];
    if (!file) return;
    readFileAsText(file, handleClientsCsv);
    clientsCsvInput.value = "";
});

teamCsvInput.addEventListener("change", () => {
    clearGlobalMessages();
    const file = teamCsvInput.files[0];
    if (!file) return;
    readFileAsText(file, handleTeamCsv);
    teamCsvInput.value = "";
});

productsCsvInput.addEventListener("change", () => {
    clearGlobalMessages();
    const file = productsCsvInput.files[0];
    if (!file) return;
    readFileAsText(file, handleProductsCsv);
    productsCsvInput.value = "";
});
```

```
    saveCheckpointBtn.addEventListener("click", () => {
        clearGlobalMessages();
        saveCheckpoint();
    });

    restoreCheckpointBtn.addEventListener("click", () => {
        clearGlobalMessages();
        restoreSelectedCheckpoint();
    });

    clearAllBtn.addEventListener("click", () => {
        clearGlobalMessages();
        clearAll();
    });

    generateReportBtn.addEventListener("click", () => {
        clearGlobalMessages();
        generateReport();
    });
});
```

Si quieres, en el siguiente paso podemos:

- Ajustar nombres de campos exactamente a tu plantilla SBS/ACFE.
- Añadir más secciones del informe (modus operandi, normas transgredidas, etc.) para que el `.doc` salga ya casi igual a tu plantilla institucional.
- Cambiar la estructura de productos para incluir montos por investigador (como “monto investigador” por colaborador) y validar que no desaparezcan columnas al hacer clic, replicando bien tu flujo real.

Y entonces otro bug que tenemos es el tema de que se repite de nuevo el pop-up de la llave técnica que debería tener un cooldown. Quise eliminarla, quizás, pensando. Como pauta general estamos poniendo que llenen las fechas en ese formato, ¿no? Sí. Bueno, es el formato que filosóficamente me gusta más. A mí también, porque no importa cómo en orden va a estar ordenado siempre, ¿no? Pero... Sí, pues, pero... Bueno, y además en LayHox es así, así que... que aguanten, que soporten. Da igual, no sé si en completado este está bien, ¿no? Porque es... Sí. Todo lo que se puede llenar... A ver, haz clic acá. Nice. Sí, este es el error de la llave técnica y otros errores de las fechas de descubrimiento. Sí, ese es el que está atorado ahí. Está atorado, sí, sí, sí, ya lo conversamos. Igual acá, esto tiene que llenar... ese autocompletado sí está bien. Sí, pero eso se marca y se pone en check, ¿no? Ya. Ya, hay que hacer algo con esto del producto, no puedo probarlo bien. Ah, sí. Eso lo voy a corregir en la tarde, pero, o sea... Ah, todo se ha duplicado el... Sí, algo pasa. Se ha duplicado la sección desplegable del collapsible, sí. Pero parece que ha quedado esta del dummy, ¿no? Que siempre está. Sí, sí, sí. Esto es como... Se ha duplicado el collapsible header, es el tema de normas, riesgos, productos, clientes, todo. Acá está tu racha de un día. Ah, sí. Ya, esto... Que también viste que hay una guía de cómo usar, si lo abres desde cero. Ah, sí, sí la he visto, que salen ahí los... ¿La has usado? ¿La has navegado? Sí, sí, pasa por cada pestaña, eso está cool. Pero se me quedó en el nueve, o sea que era algo así. ¿Así? Sí, de ahí se me volvió. Ya, entonces después hablamos de eso. Sí, porque no sé cómo sepa que vuelve a salir. Ah, pongo un riesgo y un reclamo, te acuerdas que había el tema de que se demoraba. A ver... Líder, Alessandra... Ah, mira, así se actualiza ahora en tiempo real. Nice. Nice. Pizza Party, Pizza Party. Ah, ¿y esto se verá? Este es el primer capítulo, la verdad. Sí me ha dicho que la vea, pero esto es que vea. ¿Por qué? Te vas a sentir identificado con el como que... ...fakeness del GH de Bruce Mano. Es como que, felicitaciones, me estás sacrificando tu vida. ¿Tienes un Pizza Party? No. Un carrito con pizza y prende y apaga la luz. Porque bueno, vayte chicos, ¿no? Ok, los 15 minutos de celebración han terminado. Sí, creo que sí. Exactamente. Yo estoy bastante seguro que no se puede mejorar mucho más la... ...la vieja experiencia. El Rich Text Editor. Sí... O sea, tenía que implementar un módulo adicional. Una librería aparte de esas cosas. Sí, yo las he usado, pero pues no sé qué tantas cosas externas podemos usar. Sí, te estoy diciéndolo al menos. ¿Eso funciona? ¿Puedo explotar? Guarda mi vida. Debería generar un reward. Ah, que te faltan los productos. Esto va a ser bien complicado, amigo. Es que tenemos que corregir los productos. ¿Por qué ahora tú estás en cuarenta y nueve errores? ¿Qué pasó? Ah, es que generalmente te salen... Has creado como diez productos, te acuerdas, vacíos. Hasta que empiezas a editar un producto, no te salen los warnings de cada producto. Pero si ya quieres exportar todo, ahí sí te valida todos los entries que has creado, ¿no? Sí, ya me edité todo. De tres pasó cuarenta y nueve errores. Pero hey, mira, todos los que has editado los he generado. De tu producto a tu producto. ¿Qué mide esa calidad? Tengo veinte errores. El ochenta y seis por ciento. Los campos que has llenado. Es que recuerda, la de resto de errores es porque quieres exportar y te salen los que no has editado. Pero el cálculo de todo lo demás se hace con los que tú has empezado a editar. Entonces los que has empezado a editar que no consideras esos diez productos adicionales que has creado. Está el ochenta y seis por ciento bien. Porque faltan dos campos que son los de fecha de ocurrencia, fecha de danza. Ya, pero bueno, recapitulando.

Y ahora vamos a probar con la carga vacía. Donde efectivamente se están borrando esas copias, porque... Creo que eso que he borrado no hace referencia a lo que estoy llenando ahorita. ¿Ves? Me ha salido todo de nuevo. Ya, bueno, esto. Clientes... Producto... Colaborador... Ah, y la escrolea, ¿no? Sí, la escrolea. Voy a pasar viejo. Ya, otro. Y acá. ¡Ah, ya se fue! Ya se desaparece, ¿no? ¿En algún lado? Y no está en... en la nes y el artículo se detiene. O sea, parece que sí está acá, pero estará atrás de esto. A ver, haz como que para explotar tu... Sí, no hay más. Bueno, ¿cómo vas a hacer esto? Acá está. Acá está. O sea, está debajo de algo mío. Ah, parece debajo del... O sea, está escondido debajo del... En el tab de análisis narrativo como que está invisible porque está atrás del... Del frame de análisis narrativo de los campos antecedentes, entonces está invisible. Entonces tampoco puedes interactuar. Fascinante, ¿eh? Esto de secciones extendidas, ¿qué es? Es que normalmente se llena vacío en el... Pero lo podrían llenar ahí también. ¿Qué hago? Ah, ya. Borra todos los productos, todos los... ¿no? Ah, yo no, te decía. ¿Dónde se borran efectivamente esas copias locales? Porque esto, lo que acabo de borrar... Yo, bueno, lo acabo de borrar y me ha sacado esto, ¿no? Así que debe haber alguna otra ruta en la que se esté borrando... Sí. Sí. Ya, pero después... Es que se cargó todo mío, o sea, no... No, no, pero el error es en productos. Elimínalos. Elimínalos y vamos a cargarlos por carga masiva. Y ahí debería ir a cargarse todos los campos porque se va a llenar de frente. Sí. No, no, no. Y ahora a cargar por carga masiva. Sí. Eso también lo tenías que cambiar. Eso también lo tenías que cambiar. ¿Qué cosa? Que el botón esté o arriba o abajo en ambas, o sea... Ah, sí. Si lo pones acá arriba, ponlo acá arriba. Arriba, arriba. O abajo, abajo. Ya. A ver, ahora anda a acciones. Y haz como carga masiva de productos. Eh... El producto... ¿Masivos? Ajá. Eso es verdad. Ya. La movilidad con la nación no corresponde. Sí, validación. Ah, sí. Está mal, a propósito del curso está mal. Ah, ya, ya, ya. El riesgo operacional, el riesgo de autonomía... Toma ese dinero. ¿Fue a propósito? Ajá. Luego es por mis separadores. Luego ponemos el día, ¿no? Solo me sale errores de las categorías. ¿Cuántos más productos? Ya está. ¿Qué tal? ¿Qué tal? Un resumen, ¿no? Eh, pero... O sea, acércalo... Ajá. ¿Qué tal? ¿Qué tal? ¿Qué tal? ¿Qué tal? ¿Qué tal? ¿Qué tal? Ajá. ¿Ya? Ya, ya. ¿Ya está el resultado? Bueno, han pasado muchos errores, amigo. Primero, ¿por qué hay clientes acá ahora? Eh... Resumen de clientes hay de acá. Y no sale producto. Pero si es tantos productos, ¿ah? Pero... No veo los cómodos. Ah, no, acá están. ¿No? Ya, pero acá está. Pero ¿por qué hay errores? Ya, pero sí, al... Al cargar masivo los productos... No sé si se ha pegado el catálogo de clientes en la tabla de clientes. ¿Qué ha hecho la carga de...? Claro, del catálogo, porque no hay... Ah, mostrarás clientes. No sé si ha... Ha subido... Cuando subiste carga masiva de producto, ha subido también, como si fuera... Clientes, el catálogo de clientes, ¿no? Que es incorrecto, ¿no? Si subes productos, solamente deberías subir productos y no modificar clientes. Ya sea no subir masivos, tampoco subir los... Y no vemos, en el resumen de colaboradores... A los colaboradores que sean... Deberíamos que se haya modificado, así que creemos que no hay impacto en colaboradores cuando subes producto. Pero también, ahora al subirle carga masiva de productos, no se ven en la tablita de resumen de productos. En la pestañita de productos. No se reflejan los... Productos cargados masivamente. Es como si nunca hubiesen comido galletas en su vida. Necesitaba productos. 8... Se podría trabajar. Hasta 16, ¿no? Ya, ya, ya... Ya... Si... Apago el resumen para corregir eso.

No debería pedir que tenga un colaborador, el producto. ¿Podría haber producto sin colaborador? Todas esas permutaciones deberían estar permitidas, con tal de que no se repita la llave nada más. Si yo pongo un producto, no me debería estar pidiendo que asigne a un colaborador para validar la llave. Si yo pongo el mismo producto dos veces, ahí sí debería decir, hoy se está repitiendo la llave. Pero si es el mismo producto y a uno le pongo colaborador, ahí ya no hay... Claro, o sea, podría haber el mismo producto con distintos colaboradores, pero no te debería exigir que lo pongas... Debería decir, hoy está duplicado. Diférencialo, ¿no? Porque ese mensaje que sale es como de advertencia, ¿no? Te dice, asignele un matrícula para ver si es duplicado. Ya bueno, eso es lo que debería jalar, ¿no? Que se ponga el tipo de producto, que clave cuenta... Pero esa base que armemos, van a ser consultas diferentes que voy a adaptar, ¿no? Porque tengo que sacarte las créditos, tarjetas, cuentas, todo eso. Empecemos por créditos, porque voy a sacar una muestra con esos campos. Sí, sí, no te dije que... Si puedes... ¿Te dejo una muestra grande ahorita para... No, no, no, literalmente no. Pero como que... Ah, cinco. Cinco productos, cinco tarjetas. Cinco de cada cosa sí que puedes, ¿no? Para... Cinco créditos... Cinco... Cinco cuentos, ¿no? De ahorro, ¿no? Son los principales, ¿no? Sería tarjeta de créditos y tarjeta de débito, ¿no? ¿Corrientes también tiene un formato distinto? No, no, las cuentas. O sea, las cuentas son de tablas diferentes, pero tienen un formato. Ya después de sacar cinco de cada cosa, lo voy a poner ahí, ¿no? En ese formato. Ajá. Ya, eh... No sé si con los mismos... Con los mismos... Nombres, pero bueno, todo va a ponerse acá.

Ya. Ya, pero me has comentado de más cositas también. Como por ejemplo... Como por ejemplo... Me habías dicho... Permítame. En los riesgos, ¿no? Que se podía autocompletar los riesgos. Ah, ese también necesita, ¿no? Ahí sería... De los que ya existen... Ajá. Claro, ¿no? ¿Cuál sería la estructura? La idea de riesgo sería, bueno, el RCK, ¿no? Bueno, el líder que se jala de la tabla... La descripción... Todo, de hecho. ¿Qué posición? ¿Sí? ¿Criticidad? Sí, todo se jala, en verdad. En verdad todo se jala, ¿no? Cuando ya existe el riesgo, ¿no? Ajá. Lo que haces... Lo que haces... Lo que haces... Lo que haces... Lo que haces... Lo que haces... Lo que haces... Lo que haces... Está en la base, claro, sí. Ya, el... La descripción... ¿Suele ser un campo muy grande? O sea, ¿puede ser como que... Un... Una Biblia? ¿O es como con tweet? Bueno, en fin. Si puede ser máximo un tweet. Ajá. No puede ser una Biblia, ¿no? Pero sí puede ser... Sí puede ser un tweet. Te das cuenta que estás interactuando con este tipo de cosas todo el tiempo. Ajá. Sí, eso. Y eso también te va a pasar un aviso de 0C. Sí, ¿no? Este sería un combo entonces, ¿no? O sea, ¿lo empezarías a llenar? Nuevo existente, ¿eh? A menos que en el combo pongas nuevo... Y pongas... Ay, no. Ahí debería... Claro. Deberías poder... No, siempre primero y después todos los combos. O pongas nuevo existente y que siempre... Claro, no. O... A la mala. Ponen algo que no sea... Un RCK. Ajá. Y lo considera como nuevo y ya no lo busco. O sea, ya no te va a completar. Claro. Ya. Eso. De las normas nos... O sea, las normas también hay una base, ¿no? No sé cómo han estado registrando. ¿Por qué eso no lo registramos ahorita en el script? Las normas hay un... Excel. Ya, eso sí no sé. Eso sí no sé. Lo pasan por correo. Ya. Sí, lo veo. ¿No? Ya empezó. Porque después lo de productos... ¿Algo más? Eso ya queda claro, ¿no? Reclamos. No sé si vamos a bajarnos una base de reclamos. Pega el dolor en tu hombro. No, estoy pensando. Pero para qué querría yo cruzar. Te digo solamente porque lo reclamo. Es que eso es un tema para ver qué reclamos están poniendo también. Porque ahorita ponen todos los reclamos que están asociados al caso. Pero ahorita estoy viendo reuniones que deberían poner solo el reclamo que está asociado a la pérdida. Pero... Aún no sé eso. Podríamos dejarlo todavía como texto libre, ¿no? Que lo dejen como texto porque ahí no puede ser ningún cruzo. Ya. Ya, después los fijos. ¿No? Clientes, ¿no? El cliente sí es con eco clean. Que es el IDC. No más el tipo de documento. Sí. Que es... Número de DNI... Padeado a... ¿Cuántos dígitos tiene con eco clean? ¿Veinte y tantos? Ahí es. Cero, cero, cero, cero. Más ocho dígitos. Más tipo de documento. ¿Qué tipo de documento? Uno, dos, tres, cuatro, cinco, seis, siete. Ya, ¿y tú sabes cuáles son y cuáles no? Uno es DNI. Tres es la mensajería. Seis, siete son Rooks. ¿Seis o siete son Rooks? Cuatro es pasaporte. Seis y siete son... Son Rooks. No sé exactamente qué porque. Ya. ¿No? Y me dices que para pasaporte chapan la parte que no tiene letra. Claro. Para el Rooks chapan los últimos ocho. Más documento. Más uno. El otro es cero, cero, cero. Más... Ocho dígitos. El pasaporte sí no sé exactamente cuál es. Pero creo que simplemente son los dígitos sin las letras. Dígitos sin... Ajá. Mejor más cuatro. Y el Rook es el, bueno, el... El otro es cero, cero, cero. Más... Dígitos más seis o siete. Son los ocho dígitos partiendo del tercer dígito. Ocho dígitos partiendo del tercer dígito. O sea, el dígito cuatro... Cuatro más ocho, ¿no? Porque no cuenta el tercer dígito. No cuenta, o sea, es uno, dos, y empiezas desde el tercero. Ah, y el tercero sí lo chapas. Partiendo del tercero. O sea, omites los dos primeros dígitos. Ajá. Ahí, ahí, ahí. Bueno, acá vamos... Ahorita para cuestiones de validaciones. Apellidos. Nombres, apellidos. Bueno, eso en realidad, ¿no? Teléfonos, correos. Teléfonos, correos. Separado por punto y coma. Así que lo vas a consolidar en un solo field. Lecciones, ajá. Y flag de... Pero eso no lo puedes descargar. Eso lo va a llenar el investidor. Ya, entonces eso es lo que se podría chapar, ¿no? El ID, que es el código unicoclí. Acá está el código unicoclí, código de partitura. Se descargaría, ¿no? Ajá. Ahora es con matrícula. Sería el ID. Nombres, apellidos, división. Y acá tenemos que ver, amigo, porque... No sé qué vamos a jalar en división, área y servicio, porque... Si jalamos una base, va a ser la foto actual. Ajá. Es que es imposible que nos bajemos la data histórica, amigo. O sea, es demasiada. Está en el... O sea, ya. Activos y cesados tienen también la foto actual. O sea, como que baja ahí en el colaborador. Y, o sea, solucionaríamos eso bajándonos la histórica. Sería poner fecha y matrícula y jalamos. Pero va a ser demasiado... Demasiada... Creo que podemos bajar la foto, ¿no? Por ahora, ¿no? Y en una versión futura... Sí, la foto tiene un indicio, al menos, de que sí es el colaborador. Pero lo único sería que no se sabe exactamente si es la división, el área o el puesto. Bueno, bueno. Sí, porque sería complicado estar seguro. Pero sí valdría para poner la matrícula y validar. Será por saludar. Ah, pero también hay campos de fecha de sedes. Eso sí se puede jalar. Carta renuncia. Creo que es ese, ¿no? Carta renuncia. Pero no sabemos si es eso por renuncia o por... Ajá, no. ¿Por qué no? Pero sí hay fecha de sedes. Ya. Igual que lo jalen, ¿no? O sea, igual es información que... O que no lo jalen. O sea, no me lo han pedido. Bueno, y durante la investigación no lo checan, ¿no? Procesan después. Ahí está, ¿no? Está trozado.

consider and research best practices for using the codex agent and chatgpt agent mode and prompting techniques and common errors and issues in vibe coded applications to fin possible issues and bugs and how to best prompt and create debugging agentsto try different online ways and websites to run our project and test the python tkinter gui and do different tests and checks nd validate the different frames and check ui issues nd inventory them using best testing and test documentation practices to find issues and bugs and propose fixes and double check and review the codebase to identify dependencies and possible side effects of changes to improve the propose code changes in diff style. consider as a jumping off point: "You are an expert senior Python developer and GUI specialist with deep knowledge of Tkinter. Act as a thorough code reviewer and debugger for this Python Tkinter form project and you must understand and comply with the specifications in 'Implementation_Brief_Markdown_DOCX_Report_Generation_UPDATED.md'. Your goal is to analyze the code, identify any issues, incomplete features, or bugs, preempt potential problems, provide fixes, and ensure

seamless integration across the entire codebase.
Eliminate any redundant, superfluous, or inefficient code, and flag missing components.
Follow these steps precisely:

Codebase Overview: Summarize the structure, including main modules, classes (e.g., Tkinter widgets like Entry, Button, Label), event bindings, and overall form flow. Note technologies: Python version, Tkinter imports, and any additional libraries.
Identify Issues and Bugs: Scan for common Tkinter pitfalls such as garbage collection of variables/images, threading issues causing GUI freezes, widget focusing problems, multiple Tk root instances leading to instability, timing glitches in event handlers, and security vulnerabilities (e.g., unsafe eval/exec). Also check for general Python bugs like syntax errors, logical flaws, unhandled exceptions, and performance bottlenecks. Categorize by type (e.g., UI responsiveness, data handling, security).
Feature and Functionality Verification: Verify if all form features—such as user input fields, validation, submission logic, error messages, and interactions—are completely and correctly implemented. Compare against typical form requirements [insert any specs or user stories here, e.g., 'The form should collect name, email, and submit to a file without crashes']. Highlight gaps, partial implementations, or deviations.
Integration and User Interaction Checks: Ensure all components (e.g., widgets, functions, data flows) integrate correctly. Trace user paths: from input entry to processing and output. Check for consistent styling, proper event bindings (e.g., button clicks, key presses), layout management (e.g., grid/pack), and cross-module consistency. Confirm no broken links between UI elements and backend logic.
Redundancy and Optimization: Detect duplicate code (e.g., repeated widget setups), inefficient practices (e.g., unnecessary loops in event handlers), or superfluous elements. Suggest optimizations like using StringVar for

dynamic updates or improving layout for responsiveness.
Missing Code Detection: Flag omissions such as input validation, error handling (e.g., try-except for invalid entries), documentation, unit tests for form submissions, or accessibility features.
Fixes and Improvements: For each issue, provide minimal code diffs (before → after) or snippets. Ensure fixes address Tkinter specifics, like storing image references or using after() for timing. Preempt issues by suggesting best practices (e.g., single Tk root, thread-safe updates via queue).
Final Validation and Testing: Simulate or describe tests for the updated code (e.g., run form interactions mentally or suggest test scripts). Re-review the entire revised codebase to confirm no new bugs, full integration, and optimal performance. Output the fully fixed codebase if feasible. IN all tasks that you generate you must include an instruction that before the step where you try to commit the changes and improvements as a pull request (PR) in Github you need to recopy/reimport the repo and check that there areno changes that cause conflicts in your commit so you can fix the conflicts and then create the commit and also you must double check the specifications in 'Implementation_Brief_Markdown_DOCX_Report_Generation_UPDATED.md' and ensure you comply and nothing is missing.

Respond in markdown with sections for each step. Use code blocks for snippets/diffs. Be objective, evidence-based, and prioritize user-friendly GUI design." any good prompt should include iterations steps and revalidation steps where you propose changes if possible test them and try the user interactions the generated the bug again and check the rest of the functionalities to double check no unintended side effects. Consider the different issues we have identified in this conversation as a starting list of tasks to check but you must also have the agent after double checking theses issues to do exploration and root cause analyses and find other issues and also analyze them and fix them. generate the prompt.

1. yes the public repo is: https://github.com/PillB/Formulario_investigacion 2. any. try and if it doesn't work try something else. 3. yes. we have changed definitions of functions so some of the

have changed definitions of functions so some of the sample css may be of outdated schema or content or vice versa. 4. ui and code improvements and static code analyses in that order.

You are an expert senior Python developer and GUI specialist with deep knowledge of Tkinter. Your task is to act as a thorough code reviewer and debugger for a Python Tkinter form project. The project’s code is available in the repository (repo in github: PillB/Formulario_investigacion). You must understand and comply with the specifications outlined in the "Implementation_Brief_Markdown_DOCX_Report_Generation_UPDATED.md" document (found in the repo). This specification describes required features and report-generation details that the application should fulfill. Your goal is to analyze the entire codebase, identify any issues or bugs, find incomplete features, preempt potential problems, and provide fixes and improvements. All improvements should maintain seamless integration across the codebase. Key goals: Eliminate any redundant or inefficient code, flag missing components or requirements from the spec, improve the UI/UX, and ensure robust, clean code. Prioritize UI improvements first, then code improvements, and finally static code analysis and best practices. Make sure to consider any recent changes (e.g. function definition changes that might have outdated certain UI elements or styles) when conducting the review. If some parts of the UI styling (for example, any sample CSS or theme configurations) are outdated due to recent code changes, identify and address those mismatches. Throughout the review, act systematically and iteratively: identify issues, fix them, then re-run or re-check the application to validate that the issues are resolved and no new ones are introduced. If direct execution of the Tkinter GUI is possible, attempt to run it to observe behavior (and try alternative approaches or environments if one method doesn’t work). If running the GUI in a live environment isn’t feasible, simulate user interactions and verify logic through the code. Finally, you will prepare the code for a clean commit. Before committing fixes, ensure you pull in or re-import the latest repository state to avoid conflicts, resolve any merge issues, and double-check that all specification requirements are met. Then you will create a pull request with the improvements. Output format: When responding, use markdown with clear sections for each review step below. Use bullet points or numbered lists for clarity where appropriate, and keep paragraphs concise. Include code blocks for any proposed code snippets or diffs (showing before→after changes) to illustrate fixes. Be objective and evidence-based in explanations, referencing specifics from the code. Prioritize a user-friendly GUI design in your suggestions.

1. Codebase Overview Provide a high-level summary of the project’s code structure and components: Modules and Files: List the main Python modules/files in the project (e.g. app.py, report_builder.py, and any files under ui/frames/ like team.py, risk.py, norm.py, etc.). Describe their roles (e.g. which one starts the Tkinter app, which handle different form sections or data, which generate the report). Classes and Widgets: Summarize the key classes (especially any Tkinter Frame subclasses or custom widget classes) and GUI elements. Identify major Tkinter widgets used (e.g. Tk, Toplevel, Frame, Label, Entry, Button, Text, ttk.Combobox, etc.), and how they are organized (for example, multiple frames for different tabs or steps of the form). Application Flow: Explain how the application starts and how the user navigates through it. For instance, is there a main form window (Tk() instance) that contains multiple frames or tabs? How does the app switch between different sections (e.g., “Collaborators”, “Risks”, “Norms” sections)? Describe the event flow: how user inputs are collected (e.g., clicking “Next” or “Submit” buttons), how data is stored (maybe in a CaseData or similar structure), and how the final report generation is triggered. Technologies and Libraries: Note the Python version (if specified) and any important libraries besides Tkinter (e.g. usage of csv or pandas for reading data files like team details, use of docx or markdown libraries for report generation, etc.). Mention if the app uses any external files (like a CSV for team info, or a Markdown template for the report) and how they integrate. Data Structures: Briefly mention key data structures or global variables. For example, if there is a CaseData object or dictionaries that accumulate form inputs (perhaps case_data.caso, case_data.productos, case_data.colaboradores, etc. as hinted in the spec), describe their purpose. Current State: Summarize the overall completeness of the code. Does it appear fully implemented or are some parts stubs? (E.g., maybe the UI frames exist but some functionality is not wired up yet.) This overview sets the context for deeper analysis in the next steps. 2. Identify

Issues and Bugs Thoroughly scan the code for any bugs, errors, or bad practices. Consider both Tkinter-specific pitfalls and general Python issues. Categorize and list the issues by type, providing a brief description and where in the code they occur: GUI Responsiveness & Tkinter Pitfalls: Check for common Tkinter mistakes that could cause the UI to behave incorrectly or even crash. For example: Multiple Tk Instances: Ensure only one tk.Tk() is created (multiple roots can cause instability). If the code inadvertently creates extra root windows or uses Tk() where Toplevel should be used, highlight that. Image Garbage Collection: If the app uses PhotoImage or other images/icons, verify that they are stored in variables that persist (e.g., as attributes of a class) rather than local variables that go out of scope, which would result in images not displaying GitHub

.

Threading Issues: If the code spawns threads or long-running processes (for example, for loading data or generating the report), check that Tkinter calls are confined to the main thread. Misusing threads can freeze the GUI or cause exceptions. Point out any improper use of threading or async with Tkinter.

Event Loop and Timing: Look for any use of time.sleep or long loops in the main thread that would block the GUI. If found, suggest using root.after() or background threads with queue to avoid freezing the UI.

Widget Focus and Behavior: Identify any issues with widget focus (e.g., if a Text or Entry should be focused or if navigation (Tab key order) is disrupted). Check if key events or default button behaviors are correctly handled (for instance, pressing Enter to trigger a default button).

Geometry/Display Problems: If both .pack() and .grid() are used in the same container, that’s a bug. Check for inconsistent use of geometry managers. Also ensure that UI updates (like filling data into text boxes or labels) happen on the main thread and using appropriate methods.

Data Handling & Logic Errors: Examine the code for logical bugs or exceptions that could occur during data processing:

State Management: Verify that data collected from the form (inputs in various frames) is correctly stored and passed around. For instance, ensure that after a user fills one frame and goes to the next, the data isn't lost or overwritten unintentionally. Any global state or singleton (like a CaseData object) should be handled carefully to avoid stale data between runs.

Function Calls and Definitions: Because some function definitions have changed recently (per the context, some functions' parameters or behavior might be updated), look for any function calls that no longer match the current definition (mismatched arguments or missing expected calls). These will cause runtime errors. Identify any such discrepancy (e.g., calling a function with too many/few parameters, or using a removed function).

Validation Logic: Check if the code properly validates inputs where required. For example, the spec mentions certain fields like "Centro de Costos" needing validation (digits and length)

GitHub

. If the UI allows free text but doesn't validate it, that's a functional bug. Note any missing validation or error handling for user inputs (like numeric fields, required fields left empty, etc.).

Error Handling: Look for try/except blocks or the lack thereof. Identify places where the code might throw an exception (e.g., file operations, type conversions, index errors if assuming a list element exists) but doesn't catch it, which could crash the app. Also watch for broad exception catches that just pass, which might hide issues.

Dependencies and Side Effects: Check for any hidden dependencies (e.g., reliance on an external file or a specific working directory). If the code uses relative file paths (for templates or CSVs), note if it might fail when running from a different directory. Also, if the code modifies global state or module-level variables across frames, that can introduce bugs—flag any such non-obvious coupling.

Security & Stability: Even though this is a desktop form app, ensure basic security practices:

No Unsafe Eval/Exec: Verify that the code does not use eval() or exec() on user input (or at all) — that would be a serious vulnerability and stability risk.

File Operations: If the app reads/writes files (like exporting the report or reading a CSV), ensure it sanitizes file names or paths if those come from users. Also ensure files are properly closed or used with context managers.

External Libraries: If any external library is used, ensure it's used correctly (for example, if using python-docx or openpyxl, ensure objects are saved/flushed properly).

Resource Cleanup: Check if the program gracefully handles closure (e.g., does it call root.destroy() on exit?) and if it deletes any temp files or releases any resources it opens.

Performance Bottlenecks: Identify any inefficient code that could slow down the app:

e.g., Re-reading a large CSV file repeatedly instead of once, performing heavy computations on each keystroke, or constructing large strings in inefficient ways.

Given the context, generating the report might involve iterating over many items (collaborators, products). Check if those loops are reasonable or could be optimized (though clarity is more important unless performance is truly an issue).

For each issue found, provide a clear explanation. Categorize them under headings like "UI Bug", "Logic Error", "Performance", etc. in this section for clarity. Include code references (file name and function or line) to pinpoint the problematic area. If applicable, mention how you discovered the issue (e.g., reading code or simulating an input scenario). This will set up the fixes to be discussed later.

3. Feature and Functionality Verification

Now, verify that all intended features of the form are implemented and working correctly, comparing against both common form requirements and the specific project specifications (including the Implementation Brief and any user stories or requirements known):

Intended Form Behavior: List the major features the form is supposed to provide. For example, identify all input fields and sections (from the code or spec): Case details, Collaborator details, Products involved, Risks, Norms, Conclusions, Recommendations, etc., culminating in generating an “Informe de Gerencia” report. Check each feature: is there UI for it, and does it function?

For each input field or section, verify how the code handles it. E.g., Name, Email fields example: Does the form collect the collaborator’s full name and ID correctly? In this project’s context, does selecting a collaborator ID auto-fill their name as specified

GitHub

? Is there a field for “Centro de Costos” and does it enforce the validation rules (only digits, at least 5 in a token)

GitHub

?

Check that when the user submits the form (or moves to the next section), the data is validated and either stored or an error message is shown for invalid input. If the spec or typical use case says a field is required (e.g., maybe a collaborator ID or an occurrence date), confirm the code prevents proceeding without it (or at least logs a warning).

Verify submission logic: when the user finishes filling all sections and triggers report generation, does the code gather all inputs into the report correctly? Compare against the Implementation Brief mappings: e.g., the “Referencia” in the report should summarize case info

GitHub

– does the code build this summary? If any field from the spec (like “Analítica Contable” or “Centro de Costos”) is not implemented, note it here as a missing feature.

Completeness vs Spec: Using the Implementation Brief as a checklist, go through each required report section and confirm the app provides the needed data:

Encabezado fields: Are all fields like Categoría del evento, Tipología, Importe investigado, etc., captured from user input or calculated? For instance, Importe

investigado should sum certain values
GitHub
 – ensure the code does that sum when preparing the report. Fields that the spec says are not implemented and should be left blank (like the SBS status fields
GitHub
) should be acknowledged in the code or comments; if not, flag that the app doesn't handle them (which might be okay if intended).

Collaborators table: Check if the UI collects each collaborator's name, ID (matrícula), cargo (role), etc., and if the code is prepared to list them in the report. The spec suggests auto-filling names from a CSV
GitHub
 – verify if team.py or related code does this lookup. If this feature is incomplete (e.g., name not auto-filled or CSV not loaded), note it.

Risks, Norms, Hallazgos (findings): These might be tables or lists in the report. Ensure the UI has a way to input them (maybe a list of items or multiple entry fields) and the code adds them to the report output. For example, each "hallazgo" might be a text entry; check if the user can add multiple or if it's a single text area. If the spec calls for a table but the UI only has a text box, mention the discrepancy.

Conclusions and Recommendations: These are narrative fields. Confirm that there are multi-line text boxes for them in the UI (likely in an analysis frame) and that their content goes to the report.

Report Generation: Verify that the report_builder.py (or similar) takes all the data and produces the Markdown and/or DOCX output. Check if it follows the template (perhaps Plantilla_reporte.md). If possible, ensure that after form submission, the code either saves a file or shows the report to the user. If this part is incomplete or requires manual steps (like instructing the user to copy the Markdown), highlight that.

Gaps or Incomplete Features: Identify any feature that seems partially implemented or not functioning:

Are there UI elements mentioned in the spec that are

missing in code? (e.g., a field for Fecha de reporte was needed
GitHub
 – does the UI provide a date picker or field for it? If not, that’s a missing feature).
Are there placeholders or TODO comments in code indicating unfinished parts? Mention them and what should be done.
Check if error messages are user-friendly. For instance, if validation fails, does the app show a message or just fail silently? If the latter, that’s a usability issue to fix.
If the code is supposed to write output to a file, verify that it does handle errors (like permission issues or open file). If not implemented, note it.
In this section, for each feature or requirement from the spec, clearly state whether it’s Fully Implemented, Partially Implemented, or Missing in the current code. This will guide what needs to be fixed or added.
4. Integration and User Interaction Checks
Examine how well the different parts of the application integrate and ensure a smooth user interaction flow from start to finish:
Module Integration: Verify that all modules and frames work together without breaking. For example, if app.py creates the main window and then initializes multiple frame objects (for different sections of the form), check that it properly shows/hides or switches between them. If using a system like a tkinter.Notebook or a custom controller class to switch frames, ensure each frame is initialized correctly and added to the main application. If any frame is not being displayed or updated when it should, that’s an integration issue.
Event Binding and Callbacks: Check every button or interactive widget’s command/callback. Ensure that the function it calls exists and performs the intended action: For example, a “Next” or “Submit” button should probably call a function to validate the current frame’s inputs and then either show the next frame or finalize the data. Verify those functions are connected and working. If a button has no command or calls a stub function

(pass), note that as a bug.

If keyboard events (like pressing Enter or Escape) are supposed to trigger actions (submission or closing), verify if they are bound (e.g., using <Return> events or setting a widget's default).

Pay attention to whether any callbacks rely on outdated function definitions (given the note about changed definitions). For instance, if on_submit(data) was changed to on_submit() with no parameters, but a button still does command=lambda: on_submit(form_data), it will error. Identify such mismatches.

Shared Data and State Transitions: If the user can navigate back and forth between sections, ensure that the data persists. For instance, if they go back to the "Collaborators" frame from "Risks", are their previously entered collaborator entries still there? If not, maybe the app reinitializes the frame and loses state – mention if that happens. Ideally, the design uses a single CaseData object or similar that lives throughout; verify if that's working.

User Experience Flow: Consider the overall UI/UX: Is it clear to the user how to proceed through the form? Check if each frame has a title/label indicating what section it is, and buttons like "Next", "Previous", or "Finish". If navigation is missing (e.g., no Back button where expected), suggest adding it. Ensure the final step (generating the report) gives some feedback (like a success message or opens the file) so the user knows it's done.

Consistency: Ensure consistency in design across frames (e.g., all frames use a similar style or theme, widget sizes align, consistent fonts). If one frame uses ttk.Style or a custom style and another does not, note the inconsistency. Also check that labels and messages use consistent terminology (for example, if the spec is in Spanish, the UI labels should also be in Spanish to be coherent; flag any mix of languages or inconsistent wording).

Resource Linking: Verify that external resources are properly linked. For example, if the app needs a CSV file for team info or an image for a logo, ensure the file path is correct and the file is loaded. If the path is hard-coded, that could fail on another system; suggest making it relative or configurable.

Recent Changes Impact: Given that some function definitions were changed and possibly some CSS or styling is outdated: check if any style/theme configuration (maybe a .tcl style or a separate style file) is referenced in the code. If the code expects a certain style class that isn't defined anymore due to changes, that's a broken link. For instance, if there was a function to load a style from a .json or .css-like file and it was modified, ensure the call and the file format still match. Highlight any such disconnects between code and style/data definitions.

Summarize any integration problems found and their impact on the user. For each, explain what the expected behavior is versus what the code currently does. This will set the stage for proposing fixes in a later section.

5. UI/UX Review and Improvements

Focus on the user interface design and experience to identify improvements. Even if the UI works functionally, there may be areas to refine for clarity, aesthetics, or usability:

Layout and Responsiveness: Evaluate how the form is laid out in each frame. Are widgets aligned and spaced properly? If some forms look cluttered or misaligned, suggest using options like padx, pady, or grid column spacing to improve it. Check if the window is resizable; if so, do widgets expand appropriately or do they get cut off? Recommend using responsive layouts (e.g., sticky='nsew' on grid or expand=True on pack) so the UI can resize gracefully, or if fixed size is intended, ensure it's enough to fit all content.

Widget Choice and Configuration: Identify if any widget could be better suited for input. For example, if a field expects a date, suggest using a date picker or at least a placeholder format. If a text input has a known set of options, maybe a ttk.Combobox would improve consistency. Ensure multi-line text uses Text widget with scrollbars if content can be long (e.g., for “Antecedentes” or “Conclusiones” sections). If currently just an Entry, that might be insufficient.

Visual Styling: Check if the app uses ttk themes or custom styles. If not, consider if adopting ttk widgets (which follow system theming) would make the app look more modern. If the app already uses theming (perhaps a custom CSS-like theme file as hinted), ensure that styles (colors, fonts) are applied consistently. Suggest any improvements like a consistent font across all widgets, or highlight required fields in some way (maybe bold labels or asterisk for required fields).

User Guidance and Messages: Look at what happens when the user interacts:

Are field labels descriptive enough? If not, suggest clearer phrasing or adding tooltips/help text for complex fields.

When the form is submitted successfully, the code likely shows a success message (e.g., “Report generated!”) – check if this exists and is clear. If not, recommend adding a confirmation dialog or message box indicating where the report was saved.

If the user does something wrong (like leaving a required field empty or entering invalid data), ensure the UI provides an error message dialog or at least highlights the issue. If currently the app fails silently or just prints to console, propose adding a messagebox.showerror with a friendly message to the user.

Navigation and Flow: If the form spans multiple frames, consider adding a progress indicator or steps (like “Step 2 of 5”) so the user knows how much is left. If the frames can be visited non-sequentially (maybe not in this case), ensure there’s a way to jump or a menu. Otherwise, ensure the “Next”/“Previous” navigation is foolproof (e.g., disable “Next” until required fields are filled?).

Accessibility: Although Tkinter is somewhat limited in accessibility features, mention anything obvious: for example, ensure all interactive controls can be used via keyboard (Tab navigation order should logically flow through inputs). If the app will be used in a specific language, ensure all UI text is in that language consistently. Also, consider color choices if any (avoid relying solely on color to indicate something, for color-blind users).

List out each UI/UX issue or improvement suggestion with a brief rationale. Even if not strictly “bugs,” these improvements will enhance user satisfaction and align the form with best practices. Prioritize changes that are easy wins (like label text changes or adding padding) and clearly beneficial.

6. Redundancy and Optimization

Analyze the code for any redundancy or inefficient implementation, and propose optimizations:

Duplicate Code: Look for repeated blocks of code that could be refactored. For example, if each frame file (team.py, risk.py, etc.) defines very similar widgets or functions for saving data, consider abstracting the common functionality (perhaps a base Frame class, or utility functions). Identify any obvious copy-paste sections and suggest how to consolidate them to follow DRY (Don't Repeat Yourself) principles.

Inefficient Loops or Calculations: Pinpoint any places where the code does more work than necessary. For instance, if generating the report recomputes the same totals multiple times, suggest calculating once and reusing the result. Or if the UI updates certain fields in a loop where it could update in one go, mention that. If any list or dictionary is searched repeatedly in nested loops, consider using a different data structure (like a set for membership tests).

Resource Management: Check if the code properly opens and closes resources. For example, if reading a CSV or opening a file, ensure with open(...) is used or files are closed. If the code opens a file multiple times unnecessarily, suggest reading it once and caching the data (especially relevant for the team CSV – it could be loaded at startup and reused).

Memory Footprint: In a Tkinter app, memory issues are not usually critical unless large data is stored. But if the app accumulates a large data structure (like a list of collaborators or products), and especially if the user can add many entries, ensure that those are handled efficiently. If there is any cleanup needed (like deleting widgets that are no longer shown to free memory), mention it. Also ensure images (if any) are not being duplicated unnecessarily (e.g., creating a new PhotoImage for each row when one could be reused).

Optimal Tkinter Practices: Suggest any Tkinter-specific optimizations. For example, if the code is updating the GUI in rapid succession (like adding many rows to a Frame), it might be better to batch or use .update_idletasks() appropriately. If drawing complex widgets, maybe use a Canvas if needed for performance. Only mention if relevant issues exist.

Avoiding Global Variables: If the code uses a lot of global variables or singletons, consider recommending a cleaner approach (like passing needed data as parameters or using a class to encapsulate state). Overuse of globals can lead to unintended side effects. Identify any such usage and suggest improvements (e.g., storing state in a context object or the main App class instead of module-level variables).

Commented/Dead Code: Find any sections of code that are commented out or never used. These add noise and should be removed if not needed. Flag such sections as redundant. Similarly, if there are prints or logs left from debugging, consider removing or converting them to proper logging (if logging is set up) so as not to clutter the console for end users.

For each point, clearly describe the redundant or inefficient aspect and propose a solution (e.g., “we can refactor these two similar functions into one by parameterizing X”, or “load the CSV once at startup and reuse the data, instead of reading on every submit”). If code changes are needed, you will provide snippets in the Fixes section, but you can outline the approach here.

7. Code Quality and Static Analysis

Review the code with a lens of clean code and best practices (akin to a static code analysis). This includes style, maintainability, and any warning signs of bad practice not covered above:

PEP8 Compliance: Check if the code follows standard Python style (PEP8). Are variable and function names meaningful and lowercase_with_underscores? Is the code properly indented and line-length reasonable? If you notice obvious deviations (like very long lines, inconsistent naming, missing spaces around operators, etc.), point them out. You don't need to list every minor style issue, but note patterns (e.g., "Function names are sometimes camelCase – standardize to snake_case" or "Inconsistent spacing around = in assignments.").

Comments and Documentation: Note if the code is well-commented or not. If important sections lack explanation, suggest adding comments or docstrings. If the repository includes a README or doc (which it does), ensure that the code usage matches the documentation (for instance, if README says run app.py but the main file is actually named differently, that's an issue).

Modularity and Organization: Assess how the code is organized into functions and classes. Are functions too long or doing too much? Flag any that could be split for clarity. If certain logic is embedded in the GUI code but would be better separated (MVC pattern), mention that as an improvement (though possibly a larger refactor).

Error and Exception Handling: Expand on earlier error handling by reviewing if the code handles exceptions gracefully at a high level. For example, if report generation fails (perhaps due to file I/O or bad data), does the program catch it and inform the user, or will it crash? Suggest adding try/except with user-friendly messages for critical sections.

Security Considerations: Reiterate any potential security issues discovered. For example, if the code writes to a file in a fixed location (like a temp directory), is there any risk? If it ever connects to a database or network (likely not in Tkinter offline app), ensure parameters are handled safely (no SQL injections, etc.). If not applicable, you can note that generally the app is local so risk is low, but still ensure good practices (like not leaving sensitive data in memory longer than needed, if any).

Third-Party Dependencies: Check if versions of libraries are specified or if any known issues exist. If the project relies on something like python-docx for report generation, mention if a specific version is required or if any known bug in that library might affect the app (this might be outside scope, but a quick note if relevant).
Static Analysis Tools: If you were to run a linter or tool (like flake8, pylint, or mypy for type hints), predict or identify any significant warnings they would flag. For instance, unused variables/imports, redefined functions, or potentially undefined variables. Address any you see manually (like a variable used before assignment, etc.).
Summarize the overall code quality. Is it easy to read and maintain? Highlight the positive aspects too (e.g., "The code is logically structured into modules per form section, which is good for maintainability"). Then list the areas for improvement as bullet points. The fixes for many of these (like renaming variables or adding comments) might not need detailed code snippets, but you can mention them as recommendations.
8. Missing Features or Components
After the thorough analysis, compile anything that appears to be missing or not implemented that should be there according to either the project requirements or standard form app features:
Unimplemented Spec Items: Cross-check one more time with "Implementation_Brief_Markdown_DOCX_Report_Generation_UPDATED.md". List any items that the spec/document says should be done but the code doesn't implement. For example, if the spec calls for adding fields matricula_investigador and nombre_investigador for the signature section
GitHub
and you find those are not present in the UI or data model, note that as missing. Similarly, any other "UPDATED" items in the spec (like Analítica Contable integration from reclamos, or Centro de Costos field) that are not reflected in the code should be flagged here.
Optional Enhancements: Mention features that, while not

explicitly stated, are typically expected in such applications. For example, printing or saving the report to a file (if currently it only shows on screen), or a reset/clear form functionality to allow a new entry after finishing one. If the app will be used regularly, perhaps a menu or about dialog could be nice – these might be beyond initial scope, but it's worth noting if the app feels incomplete without them.

Testing and Documentation: If the project does not include tests, highlight the absence of even basic unit tests or integration tests. For instance, functions in report_builder.py could have been unit-tested with sample data. Suggest that adding tests would help prevent regressions. Also, if the usage of the app isn't documented (e.g., how to launch, prerequisites), mention that as a missing piece in documentation.

Components Mentioned but Not Present: Sometimes code has references to things that don't exist. For example, if there's a button "Delete collaborator" but no logic implemented for it. Or a menu item that's grayed out. List such stubs or placeholders.

Future Considerations: Acknowledge any features the spec explicitly defers (like the SBS status fields to be left blank, or additional signatories). Note that they are not implemented by design, but ensure the code handles them gracefully (e.g., leaving blanks is handled, or fields are there but optional). If the code doesn't handle it (for example, there's no placeholder for those fields at all, meaning the output might be incomplete), highlight that.

Each missing feature should be clearly described, along with the impact (e.g., "User cannot input X, so the generated report will lack this information"). This helps prioritize what should be fixed now vs. noted for future.

9. Fixes and Improvements (with Code Changes)

For every issue or improvement identified in the previous sections, propose a fix. Structure this as a series of sub-points or paragraphs, each addressing a specific problem (or group of related problems), and include code snippets or minimal diffs illustrating the change. Use clear before → after formatting in code blocks for diffs.

For example:

```
# Before: Creating a new Tk instance for each frame (BAD)
window = tk.Tk()
...
window2 = tk.Tk()  # second root, problematic!

# After: Use a single Tk instance, create Toplevel or Frames for new windows/sections
root = tk.Tk()
...
top = tk.Toplevel(root)  # new window as child of root
```

For each fix, include a brief explanation of why the change is necessary (referencing the issue from earlier) and how it resolves the problem:

UI Bug Fixes: If you found specific Tkinter issues (like images not showing), provide the code solution. E.g., "Store the PhotoImage in an attribute to prevent it from garbage collection" and show the code adjustment. If navigation between frames was broken, show how to instantiate frames once and lift them instead of recreating, etc.

Logic/Function Fixes: For mismatched function calls, correct the call or the function definition so they align. Show the corrected function signature or call parameters. If input validation was missing for a field, demonstrate adding a check and perhaps using messagebox.showerror to alert the user. For example, show adding a few lines to validate "Centro de Costos" input and how to handle invalid input.

Feature Implementation: For missing features that are critical, outline how to implement them. If it's a small addition (like adding a new input field for "Centro de Costos"), show the code to add that field in the UI and how to store it in the data structure. If an entire feature is too large to fully code here, describe the approach and maybe pseudocode, focusing on integration points (so the team knows where to implement later).

Redundancy Refactor: If you suggested refactoring duplicate code, provide a snippet or two of the new consolidated function or class. For instance, show how two similar functions can be merged with a parameter. If you recommended using a loop instead of repeating code, illustrate what that loop looks like.

Optimizations: Apply any simple optimizations found. E.g., if reading the CSV on every access, modify the code to read it once and reuse it (showing the addition of a module-level cache or passing the data to where needed). If there was a time.sleep in the code, show replacing it with root.after(delay, func).

Code Quality Changes: If there were minor code style issues that could confuse future maintainers, you can show a small representative change (like renaming a confusing variable or splitting a long function into two). Ensure these changes do not alter functionality but improve clarity.

Preemptive Best Practices: Even if something hasn’t caused a bug yet, implement changes to prevent potential future issues. For example, if threads might be needed later, mention that UI updates should use thread-safe queues (just as guidance). Or if more collaborators can be added dynamically, ensure the code can handle a list of collaborators (maybe by using a loop to create widgets for each collaborator entry, instead of a fixed number of fields).

Organize the fixes in a logical order (perhaps addressing the most critical issues first, like crashing bugs, then functional issues, then refactors/improvements). Use subheadings or bullet points to separate different categories of fixes if that helps (e.g., “Crash Fix: Ensure single Tk instance”, “Feature Addition: Centro de Costos field”, “Refactor: Consolidate duplicate code in frames”). Each should have the code context and the updated code.

Make sure the code snippets are correct and consistent with the rest of the codebase. The aim is to provide a patch that, if applied, would result in a clean, working, and improved codebase aligned with all requirements.

10. Testing After Fixes

After applying the above fixes and improvements, describe how to validate that everything now works correctly and that no new issues were introduced:

Re-run the Application: Launch the updated application (or imagine doing so) and walk through the entire form as a user:

Fill out each section with sample data (including edge cases like leaving optional fields blank, entering some invalid data to trigger validations, etc.). Ensure that each validation works (e.g., try entering an invalid “Centro de Costos” like 123 to see if it shows an error, then a valid one to see it passes).

Go through navigation: click all buttons (Next, Back, Submit). Confirm that each click leads to the expected result (next frame shows up with previous data preserved, etc.).

Upon final submission, verify that the report is generated without errors. If possible, inspect the output (Markdown or DOCX) to ensure all sections are populated as expected with the sample data. For example, check that the “N° de Reclamos” field in the header shows the correct count, that blank fields are “–” where they should be blank as per spec, etc.

Regression Testing: Consider any earlier bugs and test specifically for them to be sure they are fixed:

If there was a crash when clicking a certain button before, try that again now.

If data was not saving earlier, check that workflow now.

Essentially, reproduce the scenarios of all issues from section 2 and confirm they no longer occur.

Cross-Frame Interaction: Test interactions between sections. E.g., if the user adds multiple collaborators and then maybe those should affect another section (like perhaps a summary count somewhere), verify that link. If you added the auto-fill for collaborator names, test it: entering a known ID should populate the name; entering an unknown ID should leave it blank or allow manual input as intended

GitHub

.

Edge Cases: Think of any edge cases: extremely long text in a narrative field, special characters (like accents or symbols) in inputs – does the app handle Unicode characters properly especially when generating the report? If the team CSV is missing an entry, does the app handle it? If the user tries to generate a report with no data (or cancel midway), does the app avoid crashing? Mention how the code would respond and if the fixes handled these.

Performance Check: Ensure that any optimizations didn't break functionality. If you refactored a loop or data load, test with multiple or large inputs to confirm the app still responds quickly and the outputs are correct.

User Experience Validation: Confirm that the UI/UX improvements are in effect: spacing looks better, messages appear when they should, etc. Perhaps run the app on different screen sizes (if possible) to ensure layout holds up.

No New Warnings/Errors: If you have access to run the app or at least run a linter on the code, do so after changes. Ensure no exceptions are thrown in the console during typical usage. If any new warnings or prints appear, address them.

Summarize the results of this testing phase. State that after these tests, the form works as intended for all normal use cases, the earlier identified issues are resolved, and the application meets the specifications. If any minor issues still remain (perhaps ones you plan for future or outside scope), mention them with reasoning. Otherwise, conclude that the code is now robust and ready.

11. Final Review and Commit Preparation

Before finalizing, perform a last review and take steps to integrate the changes properly into the repository:

Repository Sync: Before creating a pull request, update your local copy of the repository to the latest version. This ensures that no one else's changes will conflict with yours. If the repo has been updated in the interim (since you started editing), manually re-import or pull the latest code and merge your changes with it.

Resolve Conflicts: If any merge conflicts arise (for example, someone else edited a part of the code you also changed), carefully resolve them. Ensure that all intended fixes remain intact after merging. Test those areas again if needed.

Compliance Check: Re-read the "Implementation Brief" one more time and quickly cross-verify that every requirement marked for the current version is indeed satisfied in the code now. Make sure nothing is missing after the merge. It’s easy to lose a small change in a merge conflict resolution—double-check that, for instance, the newly added fields or validations are still present.

Code Cleanliness: Remove any debug prints or stray comments that were used during testing (unless they are helpful comments for maintainers). Ensure the code formatting is consistent (maybe run an auto-formatter if available to catch any spacing issues introduced during edits).

Commit Changes: Package all the changes into a commit. Write a clear commit message (and pull request description) summarizing the fixes and improvements. For example: “Fix GUI image loading bug, add input validation for Centro de Costos, implement auto-fill for collaborator names, refactor duplicate code in frames, and various UI/UX improvements. Ensured compliance with report spec and resolved all identified issues.”

Pull Request: Open a Pull Request on GitHub with this commit, targeting the appropriate branch (likely the main branch, unless otherwise specified). In the PR description, include a brief overview of changes, link it to any relevant issues or user stories (if the repo uses them), and mention that it has been thoroughly tested.

Request Review: Since you are acting as a senior expert, you might be the one doing the review, but if there are other team members, encourage them to review the PR. Offer to walk through the changes with them if needed (maybe not in text, but that’s the spirit).

By following these steps, you ensure the improvements integrate smoothly into the project. The final codebase should now be cleaner, more robust, and aligned with both user expectations and the formal specifications.

By adhering to all the above steps, you will have performed a comprehensive code review and delivered a set of fixes that greatly improve the Tkinter application. The form should now function correctly end-to-end, be user-friendly, and maintainable for future enhancements. All changes should be documented and committed via the PR for traceability.

Now proceed with the review and debugging process as outlined. Good luck, and ensure each step is followed diligently for the best outcome.