```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0"/>
    <title>Field Data Capture</title>

    <link rel="manifest" href="manifest.json">
    <script
src="https://cdn.sheetjs.com/xlsx-0.19.3/package/dist/xlsx.full.min.js
"></script>

    <style>
        /* ---------- BASIC LAYOUT ---------- */
        :root {
            --primary-color: #007bff;
            --secondary-color: #6c757d;
            --success-color: #28a745;
            --danger-color: #dc3545;
            --warning-color: #ffc107;
            --info-color: #17a2b8;
            --light-color: #f8f9fa;
            --dark-color: #343a40;
            --cyan-color: #0dcaf0;
        }

        body {
            font-family: -apple-system, BlinkMacSystemFont, "Segoe
UI", Roboto, "Helvetica Neue", Arial, sans-serif;
            margin: 0;
            padding: 1rem;
            background: #f2f2f2;
            color: var(--dark-color);
            -webkit-tap-highlight-color: transparent;
        }
        .main-content {
            display: none; /* Hidden by default until secret code is
entered */
        }
        h1 {
            text-align: center;
            color: #333;
            margin-top: 0.5rem;
            margin-bottom: 1.5rem;
        }

        form {
```

```
    background: #fff;
    padding: 1.5rem;
    border-radius: 8px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    display: grid;
    gap: 1rem;
    grid-template-columns: 1fr;
    max-width: 900px;
    margin: auto;
    margin-bottom: 2rem;
}
@media (min-width: 600px) {
    form {
        grid-template-columns: 1fr 1fr;
    }
}

/* ---------- FORM ELEMENTS ---------- */
.field {
    display: flex;
    flex-direction: column;
}
label {
    margin-bottom: 0.3rem;
    font-weight: 600;
    font-size: 0.9rem;
}
label .required-star {
    color: var(--danger-color);
    font-weight: bold;
    margin-left: 2px;
}
input, select, textarea {
    padding: 0.6rem;
    border: 1px solid #ccc;
    border-radius: 4px;
    font-size: 1rem;
    width: 100%;
    box-sizing: border-box;
    transition: border-color 0.2s ease, box-shadow 0.2s ease;
}
input:focus, select:focus, textarea:focus {
    outline: none;
    border-color: var(--primary-color);
    box-shadow: 0 0 0 2px rgba(0, 123, 255, 0.25);
}
textarea {
    min-height: 60px;
```

```
            resize: vertical;
        }
        input[readonly] {
            background-color: #e9e9e9;
            cursor: not-allowed;
        }

        /* ---------- BUTTONS ---------- */
        .actions {
            grid-column: 1 / -1;
            display: flex;
            gap: 1rem;
            flex-wrap: wrap;
            justify-content: center;
            margin-top: 1rem;
        }
        button {
            flex: 1 1 auto;
            min-width: 120px;
            border: none;
            border-radius: 5px;
            color: #fff;
            padding: 0.75rem 1rem;
            font-size: 1rem;
            font-weight: 500;
            cursor: pointer;
            transition: background-color 0.2s ease, transform 0.1s
ease, box-shadow 0.2s ease;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        }
        button:hover {
            opacity: 0.9;
            transform: translateY(-2px);
            box-shadow: 0 4px 8px rgba(0,0,0,0.15);
        }
        button:active {
            transform: translateY(0);
            box-shadow: 0 1px 2px rgba(0,0,0,0.1);
        }

        #gpsBtn { background: var(--success-color); }
        #submitBtn { background: var(--primary-color); }
        #viewDataBtn { background: var(--info-color); }
        #downloadOnlyBtn { background: var(--cyan-color); color: #000;
}
        #exportBtn { background: var(--warning-color); color: #000; }
        #clearDataBtn { background: var(--danger-color); }
        #closeViewDataBtn { background: var(--secondary-color); }
```

```
        /* ---------- MESSAGES & MODALS ---------- */
        #message {
            position: fixed;
            bottom: 20px;
            left: 50%;
            transform: translateX(-50%);
            background-color: rgba(0, 0, 0, 0.8);
            color: #fff;
            padding: 12px 24px;
            border-radius: 25px;
            z-index: 2000;
            opacity: 0;
            transition: opacity 0.3s ease-in-out, transform 0.3s
ease-in-out;
            pointer-events: none;
            text-align: center;
            white-space: nowrap;
            box-shadow: 0 4px 10px rgba(0,0,0,0.2);
        }
        #message.show {
            opacity: 1;
            transform: translateX(-50%) translateY(-10px);
        }

        .modal-overlay {
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            background: rgba(0, 0, 0, 0.6);
            display: flex;
            justify-content: center;
            align-items: center;
            z-index: 1500;
            opacity: 0;
            pointer-events: none;
            transition: opacity 0.3s ease;
        }
        .modal-overlay.visible {
            opacity: 1;
            pointer-events: auto;
        }
        .modal-content {
            background: #fff;
            padding: 2rem;
```

```
    border-radius: 8px;
    box-shadow: 0 5px 15px rgba(0,0,0,0.3);
    text-align: center;
    max-width: 90%;
    width: 350px;
    transform: scale(0.9);
    transition: transform 0.3s ease;
}
.modal-overlay.visible .modal-content {
    transform: scale(1);
}
.modal-content h2 { margin-top: 0; }
.modal-content input { margin: 1rem 0; }
.modal-actions {
    display: flex;
    justify-content: center;
    gap: 1rem;
}
#passwordSubmit, #confirmYesBtn {
    background-color: var(--success-color);
}
#confirmNoBtn {
    background-color: var(--secondary-color);
}

/* ---------- SAVED DATA SECTION ---------- */
#savedDataSection {
    background: #fff;
    padding: 1.5rem;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0,0,0,.1);
    max-width: 900px;
    margin: auto;
    margin-top: 2rem;
    display: none;
}
#savedDataSection.visible {
    display: block;
}
#savedDataSection h2 {
    text-align: center;
    color: #333;
    margin-top: 0;
    margin-bottom: 1rem;
}
#dataCounts {
    text-align: center;
    margin-bottom: 1rem;
```

```
            font-size: 0.95rem;
            color: var(--secondary-color);
        }
        #dataCounts span {
            font-weight: bold;
            color: var(--dark-color);
            margin: 0 10px;
        }
        #savedDataTableContainer {
            overflow-x: auto;
            max-width: 100%;
        }
        #savedDataTable {
            width: 100%;
            border-collapse: collapse;
            margin-bottom: 1rem;
        }
        #savedDataTable th, #savedDataTable td {
            border: 1px solid #ddd;
            padding: 10px;
            text-align: left;
            white-space: nowrap;
        }
        #savedDataTable th {
            background-color: #f2f2f2;
            font-weight: 600;
            position: sticky;
            top: 0;
            z-index: 2;
        }
        #savedDataTable tr:nth-child(even) {
            background-color: #f9f9f9;
        }
        #savedDataTable tr:hover {
            background-color: #f1f1f1;
        }
        .action-cell button {
            padding: 4px 8px;
            font-size: 0.8rem;
            min-width: auto;
            margin-right: 5px;
        }
        .edit-btn { background-color: var(--primary-color); }
        .delete-btn { background-color: var(--danger-color); }
    </style>
</head>
<body>
```

```
    <!-- Password Modal -->
    <div id="passwordModal" class="modal-overlay visible">
        <div class="modal-content">
            <h2>Data Encryption</h2>
            <p>Enter a password to encrypt and decrypt your data. This
password is not stored and is required every session.</p>
            <input type="password" id="passwordInput"
placeholder="Data Password">
            <div class="modal-actions">
                <button id="passwordSubmit">Unlock Data</button>
            </div>
        </div>
    </div>

    <!-- Confirmation Modal -->
    <div id="confirmModal" class="modal-overlay">
        <div class="modal-content">
            <h2 id="confirmTitle">Are you sure?</h2>
            <p id="confirmMessage"></p>
            <div class="modal-actions">
                <button id="confirmYesBtn">Yes</button>
                <button id="confirmNoBtn">No</button>
            </div>
        </div>
    </div>

    <div class="main-content">
        <h1>Field Data Capture</h1>

        <form id="captureForm" novalidate>
            <input type="hidden" id="recordId" name="id">

            <!-- Form fields remain the same -->
            <div class="field"><label for="workorderNo">Workorder
No</label><input id="workorderNo" name="workorderNo" type="text"
placeholder="Optional"></div>
            <div class="field"><label for="zone">Zone</label><input
id="zone" name="zone" type="text" placeholder="e.g., North"></div>
            <div class="field"><label
for="sector">Sector</label><input id="sector" name="sector"
type="text" placeholder="e.g., A"></div>
            <div class="field"><label for="cnc">CNC</label><input
id="cnc" name="cnc" type="text"></div>
            <div class="field"><label for="contractorName">Contractor
Name or Source</label><input id="contractorName" name="contractorName"
type="text"></div>
            <div class="field"><label for="date">Date</label><input
id="date" type="date" name="date"></div>
```

```
            <div class="field"><label for="feederName">Feeder
Name</label><input id="feederName" name="feederName"
type="text"></div>
            <div class="field"><label for="townshipName">Township
Name</label><input id="townshipName" name="townshipName"
type="text"></div>
            <div class="field"><label for="transformerNo">Transformer
No</label><input id="transformerNo" name="transformerNo"
type="text"></div>
            <div class="field"><label for="standNo">Stand
No</label><input id="standNo" name="standNo" type="text"></div>
            <div class="field"><label
for="installationNo">Installation No</label><input id="installationNo"
name="installationNo" type="text"></div>
            <!-- DEBUG FIX: Changed id and for attributes from lat/lon
to latitude/longitude to match the name attribute -->
            <div class="field"><label for="latitude">Latitude (DD MM
SS.SSS S/N)</label><input id="latitude" name="latitude" type="text"
readonly></div>
            <div class="field"><label for="longitude">Longitude (DD MM
SS.SSS E/W)</label><input id="longitude" name="longitude" type="text"
readonly></div>
            <div class="field"><label
for="access">Access</label><select id="access" name="access"><option
value="">— Select —</option><option value="Yes">Yes</option><option
value="No">No</option></select></div>
            <div class="field"><label for="atHome">At
Home</label><select id="atHome" name="atHome"><option value="">—
Select —</option><option value="Yes">Yes</option><option
value="No">No</option></select></div>
            <div class="field"><label
for="connected">Connected</label><select id="connected"
name="connected"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label
for="abandoned">Abandoned</label><select id="abandoned"
name="abandoned"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label
for="vandalised">Vandalised</label><select id="vandalised"
name="vandalised"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label
for="illegal">Illegal</label><select id="illegal"
name="illegal"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label for="owner">Owner</label><select
id="owner" name="owner"><option value="">— Select —</option><option
```

```
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label
for="surname">Surname</label><input id="surname" name="surname"
type="text"></div>
            <div class="field"><label for="name">Name</label><input
id="name" name="name" type="text"></div>
            <div class="field"><label for="idNo">ID No</label><input
id="idNo" name="idNo" type="text" inputmode="numeric" pattern="[0-9]*"
placeholder="e.g., 8501015000087"></div>
            <div class="field"><label for="phoneCode">Phone
Code</label><input id="phoneCode" name="phoneCode" type="tel"
inputmode="numeric" pattern="[0-9]*" placeholder="e.g., +27"></div>
            <div class="field"><label for="phoneNo">Phone
No</label><input id="phoneNo" name="phoneNo" type="tel"
inputmode="tel" placeholder="e.g., 821234567"></div>
            <div class="field"><label for="normalVendor">Normal
Vendor</label><select id="normalVendor" name="normalVendor"><option
value="">— Select —</option><option value="Yes">Yes</option><option
value="No">No</option></select></div>
            <div class="field"><label for="recentVendor">Most Recent
Vendor</label><input id="recentVendor" name="recentVendor"
type="text"></div>
            <div class="field"><label for="meterNo">Meter
No</label><input id="meterNo" name="meterNo" type="text"></div>
            <div class="field"><label for="split">Split</label><select
id="split" name="split"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label for="commonBaseMeter">Common/Non
Common Base Meter</label><input id="commonBaseMeter"
name="commonBaseMeter" type="text"></div>
            <div class="field"><label for="edEcu">ED/ECU</label><input
id="edEcu" name="edEcu" type="text"></div>
            <div class="field"><label for="meterType">Meter
Type</label><input id="meterType" name="meterType" type="text"></div>
            <div class="field"><label for="tariffCode">Tariff
Code</label><input id="tariffCode" name="tariffCode"
type="text"></div>
            <div class="field"><label for="ampLimit">Amp
Limit</label><input id="ampLimit" name="ampLimit" type="number"
inputmode="numeric" pattern="[0-9]*"></div>
            <div class="field"><label for="supplyGroupCode">Supply
Group Code</label><input id="supplyGroupCode" name="supplyGroupCode"
type="text"></div>
            <div class="field"><label for="magCard">Mag
Card/Keypad</label><input id="magCard" name="magCard"
type="text"></div>
            <div class="field"><label for="stsOrProp">STS or
Prop</label><input id="stsOrProp" name="stsOrProp" type="text"></div>
```

```
            <div class="field"><label
for="tampered">Tampered</label><select id="tampered"
name="tampered"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label for="tamperMethod">Tamper
Method</label><input id="tamperMethod" name="tamperMethod"
type="text"></div>
            <div class="field"><label
for="removed">Removed</label><select id="removed"
name="removed"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label for="tamperNotNo">Tamper Not
No</label><input id="tamperNotNo" name="tamperNotNo"
type="text"></div>
            <div class="field"><label for="cardTrip">Card
Trip</label><select id="cardTrip" name="cardTrip"><option value="">—
Select —</option><option value="Pass">Pass</option><option
value="Fail">Fail</option></select></div>
            <div class="field"><label for="elTest">E/L
Test</label><select id="elTest" name="elTest"><option value="">—
Select —</option><option value="Pass">Pass</option><option
value="Fail">Fail</option></select></div>
            <div class="field"><label for="meterFaulty">Meter
Faulty</label><select id="meterFaulty" name="meterFaulty"><option
value="">— Select —</option><option value="Yes">Yes</option><option
value="No">No</option></select></div>
            <div class="field"><label
for="replaced">Replaced</label><select id="replaced"
name="replaced"><option value="">— Select —</option><option
value="Yes">Yes</option><option value="No">No</option></select></div>
            <div class="field"><label for="mmfNo">MMF No</label><input
id="mmfNo" name="mmfNo" type="text"></div>
            <div class="field"><label for="newMeterNo">New Meter
No</label><input id="newMeterNo" name="newMeterNo" type="text"></div>
            <div class="field"><label for="newMeterType">New Meter
Type</label><input id="newMeterType" name="newMeterType"
type="text"></div>
            <div class="field"><label for="newMeterAmpLimit">New Meter
Amp Limit</label><input id="newMeterAmpLimit" name="newMeterAmpLimit"
type="number" inputmode="numeric" pattern="[0-9]*"></div>
            <div class="field"><label for="newMeterCardTrip">New Meter
Card Trip</label><select id="newMeterCardTrip"
name="newMeterCardTrip"><option value="">— Select —</option><option
value="Pass">Pass</option><option
value="Fail">Fail</option></select></div>
            <div class="field"><label for="newMeterElTest">New Meter
E/L Test</label><select id="newMeterElTest"
name="newMeterElTest"><option value="">— Select —</option><option
```

```
value="Pass">Pass</option><option
value="Fail">Fail</option></select></div>
            <div class="field"><label
for="sealed">Sealed</label><select id="sealed" name="sealed"><option
value="">— Select —</option><option value="Yes">Yes</option><option
value="No">No</option></select></div>
            <div class="field"><label for="sealNo">Seal
No</label><input id="sealNo" name="sealNo" type="text"></div>
            <div class="field"><label
for="remarks">Remarks</label><textarea id="remarks"
name="remarks"></textarea></div>

            <div class="actions">
                <button type="button" id="gpsBtn">Get GPS</button>
                <button type="submit" id="submitBtn">Save
Data</button>
                <button type="button" id="viewDataBtn">View
Data</button>
                <button type="button"
id="downloadOnlyBtn">Download</button>
                <button type="button" id="exportBtn">Download &
Clear</button>
            </div>
        </form>

        <div id="message" role="alert" aria-live="polite"></div>

        <div id="savedDataSection">
            <h2>Saved Field Entries</h2>
            <div id="dataCounts"></div>
            <div id="savedDataTableContainer">
                <table id="savedDataTable">
                    <thead id="savedTableHeader"></thead>
                    <tbody id="savedTableBody"></tbody>
                </table>
            </div>
            <div id="savedDataActions">
                <button type="button" id="clearDataBtn">Clear All
Saved Data</button>
                <button type="button" id="closeViewDataBtn">Close
View</button>
            </div>
        </div>
    </div>

    <script>
        const DB_NAME = 'FieldDataDB';
        const DB_VERSION = 2; // Incremented version for schema change
```

```
        const STORE_NAME = 'records';
        let db;
        let encryptionKey; // This will hold the key for the session

        // --- CRYPTOGRAPHY FUNCTIONS ---
        // Derives a key from a password using PBKDF2
        async function getKey(password, salt) {
            const enc = new TextEncoder();
            const keyMaterial = await window.crypto.subtle.importKey(
                "raw",
                enc.encode(password),
                { name: "PBKDF2" },
                false,
                ["deriveKey"]
            );
            return window.crypto.subtle.deriveKey(
                {
                    "name": "PBKDF2",
                    salt: salt,
                    "iterations": 100000,
                    "hash": "SHA-256"
                },
                keyMaterial,
                { "name": "AES-GCM", "length": 256 },
                true,
                ["encrypt", "decrypt"]
            );
        }

        // Encrypts a JSON object
        async function encryptData(dataObject) {
            if (!encryptionKey) throw new Error("Encryption key is not
set.");
            const iv = window.crypto.getRandomValues(new
Uint8Array(12)); // Initialization Vector
            const enc = new TextEncoder();
            const encodedData =
enc.encode(JSON.stringify(dataObject));

            const encryptedContent = await
window.crypto.subtle.encrypt(
                { name: "AES-GCM", iv: iv },
                encryptionKey,
                encodedData
            );

            // Combine iv and encrypted data for storage
            const encryptedPackage = {
```

```
                iv: Array.from(iv), // Convert Uint8Array to array for
JSON compatibility
                data: Array.from(new Uint8Array(encryptedContent))
            };
            return encryptedPackage;
        }

        // Decrypts an encrypted package
        async function decryptData(encryptedPackage) {
            if (!encryptionKey) throw new Error("Encryption key is not
set.");
            try {
                const iv = new Uint8Array(encryptedPackage.iv);
                const encryptedContent = new
Uint8Array(encryptedPackage.data);

                const decryptedContent = await
window.crypto.subtle.decrypt(
                    { name: "AES-GCM", iv: iv },
                    encryptionKey,
                    encryptedContent
                );

                const dec = new TextDecoder();
                return JSON.parse(dec.decode(decryptedContent));
            } catch (e) {
                console.error("Decryption failed:", e);
                throw new Error("Decryption failed. Check password or
data integrity.");
            }
        }


        // --- DATABASE FUNCTIONS ---
        function initDb() {
            return new Promise((resolve, reject) => {
                const request = indexedDB.open(DB_NAME, DB_VERSION);
                request.onupgradeneeded = (event) => {
                    db = event.target.result;
                    if (!db.objectStoreNames.contains(STORE_NAME)) {
                        db.createObjectStore(STORE_NAME, { keyPath:
'id', autoIncrement: true });
                    }
                };
                request.onsuccess = (event) => {
                    db = event.target.result;
                    resolve(db);
                };
```

```
                request.onerror = (event) =>
reject(event.target.error);
            });
        }

        function addRecord(encryptedRecord) {
            return new Promise((resolve, reject) => {
                const transaction = db.transaction([STORE_NAME],
'readwrite');
                const store = transaction.objectStore(STORE_NAME);
                // The record is now an object { id, encryptedData }
                const request = store.add({ encryptedData:
encryptedRecord });
                request.onsuccess = () => resolve();
                request.onerror = (event) =>
reject(event.target.error);
            });
        }

        function updateRecord(id, encryptedRecord) {
            return new Promise((resolve, reject) => {
                const transaction = db.transaction([STORE_NAME],
'readwrite');
                const store = transaction.objectStore(STORE_NAME);
                const request = store.put({ id: id, encryptedData:
encryptedRecord });
                request.onsuccess = () => resolve();
                request.onerror = (event) =>
reject(event.target.error);
            });
        }

        function getRecord(id) {
            return new Promise((resolve, reject) => {
                const transaction = db.transaction([STORE_NAME],
'readonly');
                const store = transaction.objectStore(STORE_NAME);
                const request = store.get(id);
                request.onsuccess = (event) =>
resolve(event.target.result);
                request.onerror = (event) =>
reject(event.target.error);
            });
        }

        // Other DB functions (delete, getAll, clearAll) remain
largely the same
        function deleteRecord(id) {
```

```
            return new Promise((resolve, reject) => {
                const transaction = db.transaction([STORE_NAME],
'readwrite');
                const store = transaction.objectStore(STORE_NAME);
                const request = store.delete(id);
                request.onsuccess = () => resolve();
                request.onerror = (event) =>
reject(event.target.error);
            });
        }

        function getAllRecords() {
            return new Promise((resolve, reject) => {
                const transaction = db.transaction([STORE_NAME],
'readonly');
                const store = transaction.objectStore(STORE_NAME);
                const request = store.getAll();
                request.onsuccess = (event) =>
resolve(event.target.result);
                request.onerror = (event) =>
reject(event.target.error);
            });
        }

        function clearAllRecords() {
            return new Promise((resolve, reject) => {
                const transaction = db.transaction([STORE_NAME],
'readwrite');
                const store = transaction.objectStore(STORE_NAME);
                const request = store.clear();
                request.onsuccess = () => resolve();
                request.onerror = (event) =>
reject(event.target.error);
            });
        }

        // --- UI FEEDBACK & MODALS ---
        const messageElement = document.getElementById('message');
        function showMessage(msg, type = 'info', duration = 3000) {
            messageElement.textContent = msg;
            messageElement.className = 'show';
            let bgColor = 'rgba(0, 0, 0, 0.8)';
            if (type === 'error') bgColor = 'rgba(220, 53, 69, 0.9)';
            else if (type === 'success') bgColor = 'rgba(40, 167, 69,
0.9)';
            messageElement.style.backgroundColor = bgColor;
            setTimeout(() => {
                messageElement.classList.remove('show');
```

```
            }, duration);
        }

        const confirmModal = document.getElementById('confirmModal');
        function showConfirm(title, message) {
            return new Promise((resolve) => {
                document.getElementById('confirmTitle').textContent =
title;
                document.getElementById('confirmMessage').textContent
= message;
                confirmModal.classList.add('visible');
                document.getElementById('confirmYesBtn').onclick = ()
=> {
                    confirmModal.classList.remove('visible');
                    resolve(true);
                };
                document.getElementById('confirmNoBtn').onclick = ()
=> {
                    confirmModal.classList.remove('visible');
                    resolve(false);
                };
            });
        }

        // --- GPS & DATE ---
        function toDMS(dec, type) {
            const abs = Math.abs(dec);
            const deg = Math.floor(abs);
            const minF = (abs - deg) * 60;
            const min = Math.floor(minF);
            const sec = ((minF - min) * 60).toFixed(3);
            let hemisphere = dec >= 0 ? (type === 'latitude' ? 'N' :
'E') : (type === 'latitude' ? 'S' : 'W');
            return `${String(deg).padStart(2, '0')}
${String(min).padStart(2, '0')} ${sec.padStart(6, '0')}
${hemisphere}`;
        }

        function setTodaysDate() {
            const today = new Date();
            document.getElementById('date').valueAsDate = today;
        }

        // --- VALIDATION ---
        const alwaysRequiredFields = ['cnc', 'date', 'feederName',
'transformerNo'];
        // DEBUG FIX: Changed 'lat', 'lon' to 'latitude', 'longitude'
to match the input names
```

```
        const conditionalRequiredFields = ['phoneCode', 'phoneNo',
'meterNo', 'meterType', 'supplyGroupCode', 'cardTrip', 'elTest',
'sealed', 'sealNo', 'latitude', 'longitude'];

        function updateRequiredVisuals() {
            const form = document.getElementById('captureForm');
            const isConditionMet = ['access', 'atHome',
'connected'].some(id => form.elements[id].value === 'Yes');
            form.querySelectorAll('.required-star').forEach(star =>
star.remove());
            const addStar = (element) => {
                const label =
element.closest('.field')?.querySelector('label');
                if (label && !label.querySelector('.required-star')) {
                    const star = document.createElement('span');
                    star.className = 'required-star';
                    star.textContent = '*';
                    label.appendChild(star);
                }
            };
            alwaysRequiredFields.forEach(id =>
addStar(form.elements[id]));
            if (isConditionMet) {
                conditionalRequiredFields.forEach(id =>
addStar(form.elements[id]));
            }
        }

        function validateForm() {
            const form = document.getElementById('captureForm');
            const data = new FormData(form);
            const errors = [];
            const isConditionMet = ['access', 'atHome',
'connected'].some(id => form.elements[id].value === 'Yes');
            const checkField = (id, label) => {
                if (!data.get(id)?.trim()) {
                    errors.push(`${label} is required.`);
                }
            };
            alwaysRequiredFields.forEach(id => {
                const label =
form.querySelector(`label[for=${id}]`).textContent.replace('*','');
                checkField(id, label);
            });
            if (isConditionMet) {
                conditionalRequiredFields.forEach(id => {
                    const label =
form.querySelector(`label[for=${id}]`).textContent.replace('*','').rep
```

```
lace(/ \(.+\)/, '');
                    checkField(id, label);
                });
            }
            if (errors.length > 0) {
                showMessage(errors.join('\n'), 'error', 5000);
                return false;
            }
            return true;
        }

        // --- FORM & DATA HANDLING (NOW WITH ENCRYPTION) ---
        async function handleFormSubmit(e) {
            e.preventDefault();
            if (!validateForm()) return;

            const form = e.target;
            const formData = new FormData(form);
            const data = Object.fromEntries(formData.entries());
            const recordId = data.id ? parseInt(data.id, 10) : null;
            delete data.id; // Don't encrypt the ID

            if (data.date) {
                const d = new Date(data.date);
                data.date = `${d.getFullYear()}/${String(d.getMonth()
+ 1).padStart(2, '0')}/${String(d.getDate()).padStart(2, '0')}`;
            }

            try {
                const encryptedData = await encryptData(data);
                if (recordId) {
                    await updateRecord(recordId, encryptedData);
                    showMessage('Data updated successfully ✔',
'success');
                } else {
                    await addRecord(encryptedData);
                    showMessage('Data saved and encrypted successfully
✔', 'success');
                }
                resetForm();
            } catch (error) {
                console.error('Failed to encrypt/save record:',
error);
                showMessage('Failed to save data. Check console for
errors.', 'error');
            }
        }
```

```
        function resetForm() {
            document.getElementById('captureForm').reset();
            document.getElementById('recordId').value = '';
            document.getElementById('submitBtn').textContent = 'Save
Data';
            setTodaysDate();
            updateRequiredVisuals();
            window.scrollTo(0, 0);
        }

        async function editEntry(id) {
            try {
                const encryptedRecord = await getRecord(id);
                if (!encryptedRecord) throw new Error("Record not
found");

                const record = await
decryptData(encryptedRecord.encryptedData);

                const form = document.getElementById('captureForm');
                for (const key in record) {
                    if (form.elements[key]) {
                        form.elements[key].value = record[key];
                    }
                }
                if (record.date) {
                    form.elements.date.value =
record.date.replace(/\//g, '-');
                }
                document.getElementById('recordId').value = id;
                document.getElementById('submitBtn').textContent =
'Update Data';
                updateRequiredVisuals();
                savedDataSection.classList.remove('visible');
                window.scrollTo(0, 0);
                showMessage('Editing entry. Scroll up to see the
form.', 'info');
            } catch (error) {
                showMessage(`Error loading data for editing:
${error.message}`, 'error');
            }
        }

        async function deleteEntry(id) {
            const confirmed = await showConfirm('Delete Entry', 'Are
you sure you want to delete this entry? This cannot be undone.');
            if (confirmed) {
                try {
```

```
                    await deleteRecord(id);
                    showMessage('Entry deleted successfully.',
'success');
                    displaySavedData();
                } catch (error) {
                    showMessage('Error deleting entry.', 'error');
                }
            }
        }

        // --- EXPORT & VIEW DATA (NOW WITH DECRYPTION) ---
        const savedDataSection =
document.getElementById('savedDataSection');
        const formFieldOrder =
Array.from(document.getElementById('captureForm').elements)
            .map(el => el.name)
            .filter((name, index, self) => name && self.indexOf(name)
=== index && name !== 'id');

        async function displaySavedData() {
            try {
                const encryptedRecords = await getAllRecords();
                if (encryptedRecords.length === 0) {
                    showMessage('No saved data to display.', 'info');
                    savedDataSection.classList.remove('visible');
                    return;
                }

                const records = [];
                for (const rec of encryptedRecords) {
                    try {
                        const decrypted = await
decryptData(rec.encryptedData);
                        decrypted.id = rec.id; // Re-attach the ID
post-decryption
                        records.push(decrypted);
                    } catch (e) {
                         showMessage('One or more records failed to
decrypt. Incorrect password?', 'error');
                         console.error("Decryption failure for record
id:", rec.id);
                         return; // Stop processing if a record fails
                    }
                }

                let tamperedCount = 0;
                let faultyCount = 0;
                records.forEach(rec => {
```

```
                    if (rec.tampered === 'Yes') tamperedCount++;
                    if (rec.meterFaulty === 'Yes') faultyCount++;
                });
                document.getElementById('dataCounts').innerHTML =
                    `Total Entries: <span>${records.length}</span> |
Tampered: <span>${tamperedCount}</span> | Meter Faulty:
<span>${faultyCount}</span>`;

                const tableHeader =
document.getElementById('savedTableHeader');
                const tableBody =
document.getElementById('savedTableBody');
                tableHeader.innerHTML = '';
                tableBody.innerHTML = '';
                const headerRow = document.createElement('tr');
                headerRow.innerHTML = '<th>#</th>';
                formFieldOrder.forEach(key => {
                    const label =
document.querySelector(`label[for="${key}"]`)?.textContent || key;
                    headerRow.innerHTML +=
`<th>${label.replace('*','')}</th>`;
                });
                headerRow.innerHTML += '<th>Actions</th>';
                tableHeader.appendChild(headerRow);

                records.forEach((record, index) => {
                    const tr = document.createElement('tr');
                    tr.innerHTML = `<td>${index + 1}</td>`;
                    formFieldOrder.forEach(key => {
                        tr.innerHTML += `<td>${record[key] ||
''}</td>`;
                    });
                    tr.innerHTML += `<td class="action-cell">
                        <button class="edit-btn"
onclick="editEntry(${record.id})">Edit</button>
                        <button class="delete-btn"
onclick="deleteEntry(${record.id})">Delete</button>
                    </td>`;
                    tableBody.appendChild(tr);
                });

                savedDataSection.classList.add('visible');
            } catch (error) {
                showMessage(`Failed to load saved data:
${error.message}`, 'error');
            }
        }
```

```
        async function performDownload() {
            const encryptedRecords = await getAllRecords();
            if (encryptedRecords.length === 0) {
                showMessage('No data to export.', 'info');
                return false;
            }

            const records = [];
            for (const rec of encryptedRecords) {
                try {
                    records.push(await
decryptData(rec.encryptedData));
                } catch(e) {
                    showMessage('Could not decrypt all data for
export. Incorrect password?', 'error');
                    return false;
                }
            }

            const map = {};
            formFieldOrder.forEach(key => {
                map[key] =
document.querySelector(`label[for="${key}"]`)?.textContent.replace('*'
,'') || key;
            });
            const formatted = records.map(rec => {
                const obj = {};
                for (const k in rec) {
                    if (k !== 'id') obj[map[k] || k] = rec[k];
                }
                return obj;
            });

            try {
                const ws = XLSX.utils.json_to_sheet(formatted, {
header: formFieldOrder.map(k => map[k]) });
                const wb = XLSX.utils.book_new();
                XLSX.utils.book_append_sheet(wb, ws, 'FieldData');
                XLSX.writeFile(wb, 'Field_Data.xlsx');
                showMessage('Data exported successfully!', 'success');
                return true;
            } catch (error) {
                showMessage('Error exporting data to Excel.',
'error');
                return false;
            }
        }
```

```
document.getElementById('downloadOnlyBtn').addEventListener('click',
async () => {
            const records = await getAllRecords();
            if (records.length === 0) return showMessage('No data to
export.', 'info');
            const confirmed = await showConfirm('Download Data', `You
have ${records.length} entries. Do you want to download the Excel
file?`);
            if (confirmed) await performDownload();
            else showMessage('Download cancelled.', 'info');
        });

        document.getElementById('exportBtn').addEventListener('click',
async () => {
            const recordsToExport = await getAllRecords();
            if (recordsToExport.length === 0) return showMessage('No
data to export.', 'info');
            const downloadConfirmed = await showConfirm('Download
Data', `You have ${recordsToExport.length} entries. Do you want to
download the Excel file?`);
            if (!downloadConfirmed) return showMessage('Download
cancelled.', 'info');

            const success = await performDownload();
            if (success) {
                const clearConfirmed = await showConfirm('Clear Data',
'Data has been downloaded. Do you want to clear all saved entries?');
                if (clearConfirmed) {
                    await clearAllRecords();
                    savedDataSection.classList.remove('visible');
                    showMessage('All saved data has been cleared.',
'success');
                }
            }
        });

        // --- INITIALIZATION & EVENT LISTENERS ---

document.getElementById('passwordSubmit').addEventListener('click',
async () => {
            const passwordInput =
document.getElementById('passwordInput');
            if (passwordInput.value.length < 8) {
                return showMessage('Password must be at least 8
characters.', 'error');
            }
```

```
            try {
                // A static salt is not ideal, but acceptable for this
offline-first use case.
                // In a real app, this would be unique per user and
stored on a server.
                const salt = new
TextEncoder().encode("a-static-salt-for-the-pwa");
                encryptionKey = await getKey(passwordInput.value,
salt);


document.getElementById('passwordModal').classList.remove('visible');
                document.querySelector('.main-content').style.display
= 'block';

                await initDb();
                setTodaysDate();
                updateRequiredVisuals();
                if (navigator.geolocation) {
                    navigator.geolocation.getCurrentPosition(
                        pos => {
                            // DEBUG FIX: Changed getElementById to
use 'latitude' and 'longitude'
                            document.getElementById('latitude').value
= toDMS(pos.coords.latitude, 'latitude');
                            document.getElementById('longitude').value
= toDMS(pos.coords.longitude, 'longitude');
                            showMessage('Initial GPS acquired.',
'success', 2000);
                            updateRequiredVisuals();
                        },
                        err => showMessage('Could not get initial
GPS.', 'info', 4000),
                        { enableHighAccuracy: true, timeout: 10000,
maximumAge: 60000 }
                    );
                }
            } catch (e) {
                showMessage('Could not initialize encryption. See
console.', 'error');
                console.error(e);
            }
        });


document.getElementById('passwordInput').addEventListener('keypress',
(e) => {
            if (e.key === 'Enter')
```

```
document.getElementById('passwordSubmit').click();
        });


document.getElementById('captureForm').addEventListener('submit',
handleFormSubmit);
        document.getElementById('gpsBtn').onclick = () => {
            if (!navigator.geolocation) return
showMessage('Geolocation not supported.', 'error');
            showMessage('Getting GPS...', 'info', 5000);
            navigator.geolocation.getCurrentPosition(
                pos => {
                    // DEBUG FIX: Changed getElementById to use
'latitude' and 'longitude'
                    document.getElementById('latitude').value =
toDMS(pos.coords.latitude, 'latitude');
                    document.getElementById('longitude').value =
toDMS(pos.coords.longitude, 'longitude');
                    showMessage('GPS acquired successfully!',
'success');
                    updateRequiredVisuals();
                },
                err => showMessage(`GPS Error: ${err.message}`,
'error'),
                { enableHighAccuracy: true, timeout: 15000,
maximumAge: 0 }
            );
        };


document.getElementById('viewDataBtn').addEventListener('click', () =>
{
             if (savedDataSection.classList.contains('visible')) {
                savedDataSection.classList.remove('visible');
            } else {
                displaySavedData();
            }
        });


document.getElementById('closeViewDataBtn').addEventListener('click',
() => {
            savedDataSection.classList.remove('visible');
        });


document.getElementById('clearDataBtn').addEventListener('click',
async () => {
```

```
            const confirmed = await showConfirm('Clear All Data', 'Are
you sure you want to clear ALL saved data? This cannot be undone.');
            if (confirmed) {
                try {
                    await clearAllRecords();
                    savedDataSection.classList.remove('visible');
                    showMessage('All saved data cleared
successfully!', 'success');
                } catch (error) {
                    showMessage('Failed to clear all data.', 'error');
                }
            }
        });

        ['access', 'atHome', 'connected'].forEach(id => {
            document.getElementById(id).addEventListener('change',
updateRequiredVisuals);
        });

    </script>
</body>
</html>
```