```python
from __future__ import annotations

from datetime import datetime, timezone
from io import BytesIO
from pathlib import Path
from uuid import uuid4
from collections import Counter, defaultdict
import hmac
import json
import math
import statistics
from html import escape
from textwrap import dedent

import qrcode
import streamlit as st
from PIL import Image

from config import get_settings
from services.flow_client import FlowClient, FlowError
from services.image_service import prepare_photo

from openai import OpenAI

import bcrypt   #for hashed passwords#


APP_VERSION = "2.0.0"
              # 1.0.0 — Initial production release
              # 1.0.1 — Small bug fix
              # 1.1.0 — New features/improvements
              # 2.0.0 — Major system upgrade

#Password generation C:\streamlittestfolder\.venv\scripts>
C:\streamlittestfolder\.venv\Scripts\python.exe -c "import bcrypt;
p=input('New password: '); print(bcrypt.hashpw(p.encode(),
bcrypt.gensalt()).decode())"

st.set_page_config(
    page_title="Equipment Maintenance",
    page_icon="🛠",
    layout="wide",
    initial_sidebar_state="expanded",
)
```

```
def apply_login_theme() -> None:
    st.markdown(
        """
        <style>
        .stApp {
            background:
                radial-gradient(circle at 90% 0%, rgba(66,186,215,.14),
transparent 28rem),
                linear-gradient(180deg, #08131d 0%, #0d1b27 100%);
            color: #f3f7fa;
        }

        [data-testid="stAppDeployButton"] {
            display: none !important;
        }

        #MainMenu { visibility: hidden; }
        footer { visibility: hidden; }
        [data-testid="stHeader"] { background: transparent; }

        [data-testid="collapsedControl"] {
            display: flex !important;
            visibility: visible !important;
            opacity: 1 !important;
            z-index: 999999 !important;
        }

        .login-panel {
            max-width: 500px;
            margin: 8vh auto 1.25rem auto;
            padding: 2rem;
            border: 1px solid #294554;
            border-radius: 1rem;
            background: linear-gradient(145deg, #142b3a, #0e1f2b);
            box-shadow: 0 15px 40px rgba(0, 0, 0, .35);
            text-align: center;
        }

        .login-title {
            color: #f3f7fa;
            font-size: 1.8rem;
            font-weight: 800;
            margin-bottom: .4rem;
        }
```

```
        .login-subtitle { color: #a8bac6; }

        [data-testid="stTextInput"] input {
            color: #f3f7fa !important;
            background: #0b1c28 !important;
            border-color: #355566 !important;
        }

        [data-testid="stWidgetLabel"] p { color: #f3f7fa !important; }

        .stButton > button, .stFormSubmitButton > button {
            color: white !important;
            background: linear-gradient(135deg, #2d8fbd, #42bad7);
            border: none;
            border-radius: .7rem;
            min-height: 2.7rem;
            font-weight: 750;
        }

        /* ============================================================
   MOBILE AND DESKTOP SELECTBOX COLOR FIX
   ============================================================ */

/* Complete selectbox container */
[data-testid="stSelectbox"] div[data-baseweb="select"] > div,
[data-testid="stSelectbox"] div[data-baseweb="select"]
div[role="combobox"] {
    background-color: #0b1c28 !important;
    color: #f3f7fa !important;
    border-color: #355566 !important;
}

/* Selected value shown inside the closed dropdown */
[data-testid="stSelectbox"] div[data-baseweb="select"] span,
[data-testid="stSelectbox"] div[data-baseweb="select"] input,
[data-testid="stSelectbox"] div[role="combobox"] span,
[data-testid="stSelectbox"] div[role="combobox"] div {
    color: #f3f7fa !important;
    -webkit-text-fill-color: #f3f7fa !important;
    opacity: 1 !important;
}

/* Dropdown arrow */
[data-testid="stSelectbox"] svg {
    fill: #f3f7fa !important;
```

```
    color: #f3f7fa !important;
}

/* Open dropdown menu */
div[data-baseweb="popover"],
div[data-baseweb="popover"] > div,
ul[role="listbox"],
div[role="listbox"] {
    background-color: #102331 !important;
    border-color: #355566 !important;
}

/* Dropdown choices */
li[role="option"],
div[role="option"] {
    background-color: #102331 !important;
    color: #f3f7fa !important;
    -webkit-text-fill-color: #f3f7fa !important;
}

/* Hovered and selected dropdown choices */
li[role="option"]:hover,
li[role="option"][aria-selected="true"],
div[role="option"]:hover,
div[role="option"][aria-selected="true"] {
    background-color: #1b4357 !important;
    color: #ffffff !important;
    -webkit-text-fill-color: #ffffff !important;
}

/* Extra protection for iPhone Safari */
@media (max-width: 768px) {
    [data-testid="stSelectbox"] div[data-baseweb="select"] > div,
    [data-testid="stSelectbox"] div[role="combobox"] {
        background: #0b1c28 !important;
        color: #f3f7fa !important;
        -webkit-text-fill-color: #f3f7fa !important;
    }

    [data-testid="stSelectbox"] * {
        opacity: 1 !important;
    }
}

#-----------------------------------------------
```

```
/* Force Streamlit select widgets to follow the dark theme */
[data-testid="stSelectbox"] [data-baseweb="select"] > div {
    background-color: #102331 !important;
    color: #f3f7fa !important;
}

[data-testid="stSelectbox"] [role="combobox"] {
    background-color: #102331 !important;
    color: #f3f7fa !important;
    -webkit-text-fill-color: #f3f7fa !important;
}

[data-testid="stSelectbox"] [role="combobox"] * {
    color: #f3f7fa !important;
    -webkit-text-fill-color: #f3f7fa !important;
}

[data-baseweb="popover"] {
    background-color: #102331 !important;
}

[data-baseweb="popover"] [role="listbox"] {
    background-color: #102331 !important;
}

[data-baseweb="popover"] [role="option"] {
    background-color: #102331 !important;
    color: #f3f7fa !important;
    -webkit-text-fill-color: #f3f7fa !important;
}
#-----------------------------------------------------------

        </style>
        """,
        unsafe_allow_html=True,
    )

#######--------------Hashed password control with user name and role------
-----------------#########

def check_password() -> bool:
    if st.session_state.get("authenticated", False):
        return True

    apply_login_theme()
```

```
login_logo = Path(__file__).parent / "Assets" / "company_logo.jpg"
if login_logo.exists():
    st.image(str(login_logo), width=135)

st.markdown(
    """
    <div class="login-panel">
        <div class="login-title">🔐 Equipment Maintenance</div>
        <div class="login-subtitle">
            Enter your authorized username and password.
        </div>
    </div>
    """,
    unsafe_allow_html=True,
)

with st.form("login_form"):
    username = st.text_input(
        "Username",
        placeholder="Enter username",
    )

    password = st.text_input(
        "Password",
        type="password",
        placeholder="Enter password",
    )

    login_clicked = st.form_submit_button(
        "Log In",
        type="primary",
        use_container_width=True,
    )

if not login_clicked:
    return False

username_key = username.strip().lower()

try:
    users = st.secrets["users"]
except KeyError:
    st.error(
        "User accounts are not configured in secrets.toml."
```

```
    )
    return False

if username_key not in users:
    st.error("Incorrect username or password.")
    return False

user_record = users[username_key]

password_hash = str(
    user_record.get("password_hash", "")
)

active = bool(
    user_record.get("active", False)
)

if not active or not password_hash:
    st.error("Incorrect username or password.")
    return False

try:
    password_valid = bcrypt.checkpw(
        password.encode("utf-8"),
        password_hash.encode("utf-8"),
    )
except ValueError:
    st.error("Invalid password configuration.")
    return False

if not password_valid:
    st.error("Incorrect username or password.")
    return False

role = str(
    user_record.get("role", "viewer")
).strip().lower()

display_name = str(
    user_record.get(
        "display_name",
        username_key,
    )
).strip()
```

```
    st.session_state["authenticated"] = True
    st.session_state["username"] = username_key
    st.session_state["display_name"] = display_name
    st.session_state["role"] = role

    st.rerun()

    return True

##########------------------------end hashed password control-------------
-------#############

BASE_DIR = Path(__file__).parent
LOGO = BASE_DIR / "Assets" / "company_logo.jpg"

if LOGO.exists():
    logo = Image.open(LOGO)
    col1, col2 = st.columns([1, 7])
    with col1:
        st.image(logo, width=90)
    with col2:
        st.markdown(
            """
            <h1 style="color:white; margin-bottom:0;">
                Equipment Maintenance Management System
            </h1>
            <p style="color:#BFC9CA;">
                Welcome to Earthrise's first AI enhanced APP, fully
internally integraded and customized for: Predictive • Prescriptive •
Preventive • Corrective Maintenance
            </p>
            """,
            unsafe_allow_html=True,
        )
else:
    st.warning(f"Logo was not found at: {LOGO}")


st.markdown(
    """
    <style>
    [data-testid="stAppDeployButton"] { display: none !important; }
    #MainMenu { visibility: hidden; }
    footer { visibility: hidden; }
    [data-testid="stHeader"] { background: transparent; }
```

```
    [data-testid="collapsedControl"] {
        display: flex !important;
        visibility: visible !important;
        opacity: 1 !important;
        z-index: 999999 !important;
    }
    </style>
    """,
    unsafe_allow_html=True,
)


ACTIVE_STATUSES = {
    "Draft",
    "Reported",
    "Assigned",
    "Repair in Progress",
    "Waiting for Parts",
    "Waiting for Contractor",
    "Awaiting Operational Release",
}

STATUS_STEPS = [
    "Reported",
    "Assigned",
    "Repair in Progress",
    "Awaiting Operational Release",
    "Closed - Returned to Operation",
]


ROLE_PAGES = {
    "viewer": [
        "Dashboard",
        "Equipment History",
        "AI Assistant",
    ],

    "reporter": [
        "Dashboard",
        "Report Breakdown",
        "Equipment History",
        "Generate QR Code",
        "AI Assistant",
    ],
```

```
    "technician": [
        "Dashboard",
        "Report Breakdown",
        "Maintenance Queue",
        "Equipment History",
        "Generate QR Code",
        "AI Assistant",
    ],

    "manager": [
        "Dashboard",
        "Report Breakdown",
        "Maintenance Queue",
        "Operational Release",
        "Equipment History",
        "Add Equipment",
        "Generate QR Code",
        "AI Assistant",
    ],
}

def current_role() -> str:
    return str(
        st.session_state.get("role", "viewer")
    ).strip().lower()


def current_username() -> str:
    return str(
        st.session_state.get("username", "")
    ).strip()


def current_display_name() -> str:
    return str(
        st.session_state.get("display_name", "")
    ).strip()


def require_roles(*allowed_roles: str) -> None:
    role = current_role()

    allowed = {
        item.strip().lower()
```

```
        for item in allowed_roles
    }

    if role not in allowed:
        st.error(
            "⛔ You do not have permission to access this page."
        )
        st.stop()


def apply_theme() -> None:
    st.markdown(
        """
        <style>
        :root {
            --bg: #08131d;
            --bg-2: #0d1b27;
            --sidebar: #07111a;
            --surface: #102331;
            --surface-2: #142b3a;
            --surface-3: #193445;
            --blue: #2d8fbd;
            --cyan: #42bad7;
            --green: #46b985;
            --amber: #e5ad46;
            --red: #ef7070;
            --text: #f3f7fa;
            --muted: #a8bac6;
            --line: #294554;
            --shadow: rgba(0, 0, 0, .32);
        }

        html, body, [class*="css"] {
            color: var(--text);
        }

        .stApp {
            color: var(--text);
            background:
                radial-gradient(circle at 95% 0%, rgba(66,186,215,.13),
transparent 28rem),
                radial-gradient(circle at 0% 100%, rgba(45,143,189,.10),
transparent 30rem),
                linear-gradient(180deg, var(--bg) 0%, var(--bg-2) 100%);
        }
```

```
[data-testid="stToolbar"] {
    color: var(--text);
}


/* SIDEBAR — DESKTOP AND MOBILE */
[data-testid="stSidebar"] {
    background:
        linear-gradient(
            180deg,
            #07111a 0%,
            #0b1c28 55%,
            #102331 100%
        ) !important;
    border-right: 1px solid #294554 !important;
}

[data-testid="stSidebar"] > div:first-child {
    background:
        linear-gradient(
            180deg,
            #07111a 0%,
            #0b1c28 55%,
            #102331 100%
        ) !important;
}

/* Make all sidebar text readable */
[data-testid="stSidebar"] *,
[data-testid="stSidebar"] p,
[data-testid="stSidebar"] span,
[data-testid="stSidebar"] label,
[data-testid="stSidebar"] div {
    color: #f3f7fa !important;
}

/* Sidebar title */
[data-testid="stSidebar"] h1,
[data-testid="stSidebar"] h2,
[data-testid="stSidebar"] h3 {
    color: #ffffff !important;
}
```

```
/* Sidebar caption */
[data-testid="stSidebar"] [data-testid="stCaptionContainer"],
[data-testid="stSidebar"] [data-testid="stCaptionContainer"] p {
   color: #a8bac6 !important;
}

/* Navigation rows */
[data-testid="stSidebar"] .stRadio label {
    padding: .55rem .65rem;
    border-radius: .7rem;
    background: transparent !important;
}

/* Navigation hover */
[data-testid="stSidebar"] .stRadio label:hover {
    background: rgba(66, 186, 215, .14) !important;
}

/* Radio text */
[data-testid="stSidebar"] .stRadio label p {
    color: #f3f7fa !important;
    font-weight: 600 !important;
}

/* Selected radio circle */
[data-testid="stSidebar"] input[type="radio"]:checked + div {
    color: #42bad7 !important;
}

/* Divider */
[data-testid="stSidebar"] hr {
    border-color: #355566 !important;
}

/* Logout button */
[data-testid="stSidebar"] .stButton > button {
    color: #ffffff !important;
    background: #18384a !important;
    border: 1px solid #3f6477 !important;
}

[data-testid="stSidebar"] .stButton > button:hover {
    background: #214d64 !important;
    border-color: #58a9c6 !important;
}
```

```
        .block-container {
            max-width: 1500px;
            padding-top: 1.25rem;
            padding-bottom: 3rem;
        }

        h1, h2, h3, h4, h5, h6,
        p, li, label, span, div {
            color: var(--text);
        }

        .stMarkdown, .stCaption, [data-testid="stCaptionContainer"] {
            color: var(--text);
        }

        [data-testid="stWidgetLabel"] p,
        [data-testid="stWidgetLabel"] label,
        [data-testid="stWidgetLabel"] span {
            color: var(--text) !important;
            font-weight: 650;
        }

        .app-header {
            display: flex;
            align-items: center;
            justify-content: space-between;
            gap: 1rem;
            padding: 1.25rem 1.4rem;
            margin-bottom: 1.2rem;
            border-radius: 1rem;
            color: white;
            border: 1px solid rgba(89,184,218,.28);
            background:
                linear-gradient(120deg, rgba(18,50,74,.98),
rgba(31,111,159,.94)),
                repeating-linear-gradient(
                    -45deg,
                    rgba(255,255,255,.04),
                    rgba(255,255,255,.04) 10px,
                    transparent 10px,
                    transparent 20px
                );
            box-shadow: 0 14px 34px var(--shadow);
        }
```

```
        .app-header * {
            color: white !important;
        }

        .app-header-title {
            font-size: 1.65rem;
            line-height: 1.1;
            font-weight: 800;
        }

        .app-header-subtitle {
            margin-top: .35rem;
            color: rgba(255,255,255,.78) !important;
            font-size: .92rem;
        }

        .header-chip {
            white-space: nowrap;
            border: 1px solid rgba(255,255,255,.24);
            background: rgba(255,255,255,.10);
            border-radius: 999px;
            padding: .5rem .8rem;
            font-size: .8rem;
            font-weight: 700;
        }

        .section-card,
        .equipment-card,
        .kpi-card {
            border: 1px solid var(--line);
            background: linear-gradient(145deg, rgba(20,43,58,.98),
rgba(14,31,43,.98));
            box-shadow: 0 10px 28px var(--shadow);
        }

        .section-card {
            border-radius: 1rem;
            padding: 1rem 1.1rem;
            margin-bottom: 1rem;
        }

        .section-card *,
        .equipment-card *,
        .kpi-card * {
```

```
    color: var(--text) !important;
}

.kpi-card {
    min-height: 116px;
    padding: 1rem 1.05rem;
    border-radius: .95rem;
}

.kpi-label {
    color: var(--muted) !important;
    font-size: .78rem;
    font-weight: 750;
    text-transform: uppercase;
    letter-spacing: .06em;
}

.kpi-value {
    color: var(--text) !important;
    font-size: 1.8rem;
    font-weight: 850;
    margin-top: .35rem;
}

.kpi-note,
.equipment-meta {
    color: var(--muted) !important;
}

.kpi-note {
    font-size: .78rem;
    margin-top: .2rem;
}

.status-badge {
    display: inline-flex;
    align-items: center;
    gap: .4rem;
    padding: .35rem .68rem;
    border-radius: 999px;
    font-size: .78rem;
    font-weight: 800;
    border: 1px solid transparent;
}
```

```
        .status-reported { color: #ffd5ad !important; background: #593018;
border-color: #89502a; }
        .status-assigned { color: #bfeaff !important; background: #173d54;
border-color: #2e6988; }
        .status-progress { color: #ffe49a !important; background: #4c3e16;
border-color: #7d6826; }
        .status-waiting { color: #ffd994 !important; background: #503716;
border-color: #825b25; }
        .status-closed { color: #b8f1d7 !important; background: #183f31;
border-color: #2d7055; }
        .status-default { color: #d6e0e6 !important; background: #293b47;
border-color: #496171; }

        .equipment-card {
            border-radius: 1rem;
            padding: 1.05rem 1.15rem;
            margin-bottom: 1rem;
        }

        .equipment-name {
            font-size: 1.2rem;
            font-weight: 850;
            color: var(--text) !important;
        }

        .equipment-meta {
            font-size: .85rem;
            margin-top: .25rem;
        }

        .timeline {
            display: grid;
            grid-template-columns: repeat(5, minmax(100px, 1fr));
            gap: .55rem;
            margin: .75rem 0 1.1rem 0;
        }

        .timeline-step {
            position: relative;
            text-align: center;
            border-radius: .75rem;
            padding: .65rem .5rem;
            border: 1px solid var(--line);
            background: #132734;
            color: #9fb0ba !important;
```

```
    font-size: .76rem;
    font-weight: 750;
}

.timeline-step.done {
    color: #b8f1d7 !important;
    background: #183f31;
    border-color: #2d7055;
}

.timeline-step.current {
    color: #d8f5ff !important;
    background: #173d54;
    border-color: #2e8db1;
    box-shadow: inset 0 0 0 1px rgba(66,186,215,.16);
}

div[data-testid="stForm"] {
    border: 1px solid var(--line);
    border-radius: 1rem;
    padding: 1rem 1rem .3rem 1rem;
    background: rgba(16,35,49,.92);
    box-shadow: 0 10px 28px var(--shadow);
}

div[data-baseweb="input"] > div,
div[data-baseweb="textarea"] > div,
div[data-baseweb="select"] > div,
[data-testid="stNumberInput"] input,
[data-testid="stTextInput"] input,
[data-testid="stTextArea"] textarea {
    color: var(--text) !important;
    background: #0b1c28 !important;
    border-color: #355566 !important;
}

input, textarea {
    color: var(--text) !important;
    caret-color: var(--cyan) !important;
}

input::placeholder,
textarea::placeholder {
    color: #78909e !important;
    opacity: 1;
```

```
}

div[data-baseweb="select"] span,
div[data-baseweb="select"] input {
    color: var(--text) !important;
}

ul[role="listbox"],
div[role="listbox"] {
    background: #102331 !important;
    border: 1px solid var(--line) !important;
}

li[role="option"] {
    color: var(--text) !important;
    background: #102331 !important;
}

li[role="option"]:hover,
li[aria-selected="true"] {
    background: #1b4357 !important;
}

[data-testid="stFileUploaderDropzone"] {
    background: #0d202d;
    border-color: #355566;
}

[data-testid="stFileUploaderDropzone"] * {
    color: var(--text) !important;
}

[data-testid="stExpander"] {
    background: rgba(16,35,49,.92);
    border: 1px solid var(--line);
    border-radius: .85rem;
}

[data-testid="stExpander"] summary,
[data-testid="stExpander"] summary * {
    color: var(--text) !important;
}

.stButton > button,
.stDownloadButton > button,
```

```
.stLinkButton > a,
.stFormSubmitButton > button {
    color: var(--text) !important;
    background: #18384a;
    border: 1px solid #3f6477;
    border-radius: .7rem !important;
    min-height: 2.7rem;
    font-weight: 750;
}

.stButton > button:hover,
.stDownloadButton > button:hover,
.stLinkButton > a:hover,
.stFormSubmitButton > button:hover {
    background: #214d64;
    border-color: #58a9c6;
}

.stButton > button[kind="primary"],
.stFormSubmitButton > button[kind="primary"] {
    color: white !important;
    background: linear-gradient(135deg, var(--blue), var(--cyan));
    border: none;
    box-shadow: 0 7px 16px rgba(45,143,189,.28);
}

button:disabled,
button[disabled] {
    color: #71818c !important;
    background: #17252e !important;
    border-color: #2b3c47 !important;
    opacity: .75;
}

[data-testid="stMetric"] {
    border: 1px solid var(--line);
    background: rgba(16,35,49,.94);
    border-radius: .85rem;
    padding: .75rem .9rem;
    box-shadow: 0 8px 22px var(--shadow);
}

[data-testid="stMetric"] * {
    color: var(--text) !important;
}
```

```
[data-testid="stAlert"] {
    border: 1px solid rgba(255,255,255,.12);
    color: var(--text);
}

[data-testid="stAlert"] * {
    color: inherit !important;
}

[data-testid="stDataFrame"] {
    border: 1px solid var(--line);
    border-radius: .8rem;
    overflow: hidden;
}

[data-testid="stDataFrame"] * {
    color: var(--text);
}

code, pre {
    color: #d9f5ff !important;
    background: #071722 !important;
    border-color: var(--line) !important;
}

hr {
    border-color: var(--line) !important;
}

a {
    color: #7ed8f0;
}

@media (max-width: 900px) {
    .app-header {
        align-items: flex-start;
        flex-direction: column;
    }
    .timeline {
        grid-template-columns: 1fr;
    }
    .header-chip {
        white-space: normal;
    }
```

```
        }
        </style>
        """,
        unsafe_allow_html=True,
    )

def render_header(section: str, subtitle: str) -> None:
    st.markdown(
        f"""
        <div class="app-header">
            <div>
                <div class="app-header-title">🛠️ Equipment
Maintenance</div>
                <div class="app-header-subtitle">{section} ·
{subtitle}</div>
            </div>
            <div class="header-chip">QR · Power Automate · SharePoint ·
Power BI</div>
        </div>
        """,
        unsafe_allow_html=True,
    )


def field_value(value):
    """Return a plain value when Power Automate sends a SharePoint field
object."""
    if isinstance(value, dict):
        return (
            value.get("Value")
            or value.get("value")
            or value.get("Url")
            or value.get("url")
            or ""
        )
    return value

def parse_utc_datetime(value):
    """
    Convert a SharePoint/Power Automate UTC date into a Python datetime.
    Returns None when the value is blank or invalid.
    """
    value = field_value(value)

    if not value:
```

```
        return None

    try:
        text = str(value).strip()

        if text.endswith("Z"):
            text = text[:-1] + "+00:00"

        parsed = datetime.fromisoformat(text)

        if parsed.tzinfo is None:
            parsed = parsed.replace(tzinfo=timezone.utc)

        return parsed.astimezone(timezone.utc)

    except (TypeError, ValueError):
        return None


def maintenance_time_invested_minutes(event: dict):
    """
    Calculate elapsed time from breakdown report to repair completion.

    Start:
        ReportedDateTimeUTC

    End:
        RepairCompletedUTC
    """
    reported_time = parse_utc_datetime(
        event.get("ReportedDateTimeUTC")
    )

    completed_time = parse_utc_datetime(
        event.get("RepairCompletedUTC")
    )

    if reported_time is None or completed_time is None:
        return None

    minutes = (
        completed_time - reported_time
    ).total_seconds() / 60

    return max(0.0, round(minutes, 1))
```

```python
def format_duration(minutes):
    """
    Display maintenance time in a readable format.
    """
    if minutes is None:
        return "Not available"

    total_minutes = int(round(minutes))
    days, remainder = divmod(total_minutes, 1440)
    hours, remaining_minutes = divmod(remainder, 60)

    parts = []

    if days:
        parts.append(f"{days} d")

    if hours:
        parts.append(f"{hours} hr")

    if remaining_minutes or not parts:
        parts.append(f"{remaining_minutes} min")

    return " ".join(parts)


#-------------These functions prevent the AI analytics page from crashing when SharePoint contains blank values, text-formatted numbers, or field objects

def safe_text(value, default=""):
    """
    Convert SharePoint or Power Automate values into readable text.
    """
    cleaned = field_value(value)

    if cleaned is None:
        return default

    cleaned = str(cleaned).strip()

    return cleaned if cleaned else default


def safe_float(value, default=0.0):
```

```
    """
    Convert SharePoint numeric fields into a float safely.
    """
    cleaned = field_value(value)

    if cleaned in (None, ""):
        return default

    try:
        return float(cleaned)
    except (TypeError, ValueError):
        return default


def safe_bool(value):
    """
    Convert SharePoint Boolean-style values to True or False.
    """
    cleaned = field_value(value)

    if isinstance(cleaned, bool):
        return cleaned

    return str(cleaned).strip().lower() in {
        "true",
        "yes",
        "1",
        "y",
    }


def parse_datetime(value):
    """
    Convert an ISO timestamp from Power Automate into a Python datetime.
    """
    cleaned = field_value(value)

    if not cleaned:
        return None

    try:
        text = str(cleaned).strip()

        if text.endswith("Z"):
            text = text[:-1] + "+00:00"
```

```
            parsed = datetime.fromisoformat(text)

            if parsed.tzinfo is None:
                parsed = parsed.replace(tzinfo=timezone.utc)

            return parsed.astimezone(timezone.utc)

        except (TypeError, ValueError):
            return None
# ------------------------------End of functions safety to prevent AI
collapse--------------

#####-------below function is to normalize the maintenance records--------
-------------#########
def normalize_maintenance_event(event: dict) -> dict:
    """
    Convert one raw Power Automate maintenance event into a predictable
    dictionary for calculations and AI analysis.
    """

    reported_datetime = parse_datetime(
        event.get("ReportedDateTimeUTC")
        or event.get("ReportedDateTime")
        or event.get("Created")
    )

    released_datetime = parse_datetime(
        event.get("ReleasedDateTimeUTC")
        or event.get("ReleasedDateTime")
    )

    repair_completed_datetime = parse_datetime(
        event.get("RepairCompletedUTC")
        or event.get("RepairCompletedDateTimeUTC")
        or event.get("RepairCompletedDateTime")
    )

    maintenance_start_datetime = parse_datetime(
        event.get("MaintenanceStartUTC")
        or event.get("MaintenanceStartDateTimeUTC")
        or event.get("MaintenanceStartDateTime")
    )

    maintenance_time_invested_minutes = None
```

```
if (
    reported_datetime is not None
    and repair_completed_datetime is not None
):
    maintenance_time_invested_minutes = max(
        0.0,
        (
            repair_completed_datetime
            - reported_datetime
        ).total_seconds() / 60,
    )

downtime_minutes = safe_float(
    event.get("DowntimeMinutes"),
    default=0.0,
)

maintenance_time_minutes = None

if (
    reported_datetime is not None
    and repair_completed_datetime is not None
):
    maintenance_time_minutes = max(
        0.0,
        (
            repair_completed_datetime
            - reported_datetime
        ).total_seconds() / 60,
    )

repair_minutes = safe_float(
    event.get("RepairMinutes"),
    default=0.0,
)

# Calculate downtime when the stored value is blank.
if (
    downtime_minutes <= 0
    and reported_datetime is not None
    and released_datetime is not None
):
    downtime_minutes = max(
        0.0,
```

```
            (
                released_datetime - reported_datetime
            ).total_seconds() / 60,
        )

    # Calculate repair time when the stored value is blank.
    if (
        repair_minutes <= 0
        and maintenance_start_datetime is not None
        and repair_completed_datetime is not None
    ):
        repair_minutes = max(
            0.0,
            (
                repair_completed_datetime
                - maintenance_start_datetime
            ).total_seconds() / 60,
        )

    return {
        "event_id": safe_text(event.get("EventID"), "Unknown"),
        "equipment_id": safe_text(
            event.get("EquipmentID"),
            "Unknown",
        ),
        "equipment_name": safe_text(
            event.get("EquipmentName"),
            "Unknown equipment",
        ),
        "department": safe_text(
            event.get("Department"),
            "Unknown",
        ),
        "location": safe_text(
            event.get("Location"),
            "Unknown",
        ),
        "status": safe_text(
            event.get("Status"),
            "Unknown",
        ),
        "priority": safe_text(
            event.get("Priority"),
            "Unknown",
        ),
```

```
"event_type": safe_text(
    event.get("EventType"),
    "Unknown",
),
"failure_category": safe_text(
    event.get("FailureCategory"),
    "Unknown",
),
"failure_mode": safe_text(
    event.get("FailureMode"),
    "Not recorded",
),
"problem_description": safe_text(
    event.get("ProblemDescription"),
    "Not recorded",
),
"root_cause": safe_text(
    event.get("RootCause"),
    "Not recorded",
),
"corrective_action": safe_text(
    event.get("CorrectiveAction"),
    "Not recorded",
),
"parts_used": safe_text(
    event.get("PartsUsed"),
    "Not recorded",
),
"follow_up_required": safe_bool(
    event.get("FollowUpRequired")
),
"follow_up_description": safe_text(
    event.get("FollowUpDescription"),
    "Not recorded",
),
"production_stopped": safe_bool(
    event.get("ProductionStopped")
),
"safety_concern": safe_bool(
    event.get("SafetyConcern")
),
"quality_concern": safe_bool(
    event.get("QualityConcern")
),
"counts_as_failure": safe_bool(
```

```
            event.get("CountsAsFailure")
        ),
        "reported_datetime": reported_datetime,
        "released_datetime": released_datetime,

        "maintenance_time_invested_minutes": (
            round(maintenance_time_invested_minutes, 2)
            if maintenance_time_invested_minutes is not None
            else None
        ),

        "downtime_minutes": round(downtime_minutes, 2),
        "repair_minutes": round(repair_minutes, 2),

    }

##### ----------------------end of maintenance-record normalizer----------
-----------------##########

########-------------below code is to add a function that calculates
reliability metrics-------------#########
def calculate_reliability_metrics(events: list[dict]) -> dict:
    """
    Calculate maintenance and reliability metrics before data is sent
    to the AI.
    """

    normalized = [
        normalize_maintenance_event(event)
        for event in events
    ]

    real_events = [
        event
        for event in normalized
        if event["status"] != "Draft"
    ]

    failure_events = [
        event
        for event in real_events
        if event["counts_as_failure"]
        or event["event_type"].lower() == "breakdown"
    ]
```

```
closed_events = [
    event
    for event in real_events
    if event["status"] == "Closed - Returned to Operation"
]

open_events = [
    event
    for event in real_events
    if event["status"] in ACTIVE_STATUSES
    and event["status"] != "Draft"
]

downtime_values = [
    event["downtime_minutes"]
    for event in closed_events
    if event["downtime_minutes"] > 0
]

maintenance_time_values = [
    event["maintenance_time_invested_minutes"]
    for event in real_events
    if event["maintenance_time_invested_minutes"] is not None
]
equipment_groups = defaultdict(list)

for event in real_events:
    equipment_key = (
        event["equipment_id"],
        event["equipment_name"],
    )
    equipment_groups[equipment_key].append(event)

equipment_metrics = []

for (
    equipment_id,
    equipment_name,
), equipment_events in equipment_groups.items():

    equipment_failures = [
        event
        for event in equipment_events
        if event["counts_as_failure"]
        or event["event_type"].lower() == "breakdown"
```

```
]

equipment_closed = [
    event
    for event in equipment_events
    if event["status"]
    == "Closed - Returned to Operation"
]

equipment_downtime = sum(
    event["downtime_minutes"]
    for event in equipment_closed
)

equipment_repair_values = [
    event["repair_minutes"]
    for event in equipment_closed
    if event["repair_minutes"] > 0
]

failure_dates = sorted(
    [
        event["reported_datetime"]
        for event in equipment_failures
        if event["reported_datetime"] is not None
    ]
)

mtbf_hours = None

if len(failure_dates) >= 2:
    gaps = [
        (
            failure_dates[index]
            - failure_dates[index - 1]
        ).total_seconds() / 3600
        for index in range(1, len(failure_dates))
    ]

    positive_gaps = [
        gap
        for gap in gaps
        if gap >= 0
    ]
```

```
    if positive_gaps:
        mtbf_hours = round(
            statistics.mean(positive_gaps),
            2,
        )

categories = Counter(
    event["failure_category"]
    for event in equipment_failures
    if event["failure_category"] != "Unknown"
)

root_causes = Counter(
    event["root_cause"]
    for event in equipment_failures
    if event["root_cause"] != "Not recorded"
)

open_count = sum(
    1
    for event in equipment_events
    if event["status"] in ACTIVE_STATUSES
    and event["status"] != "Draft"
)

critical_count = sum(
    1
    for event in equipment_events
    if event["priority"] == "Critical"
)

equipment_metrics.append(
    {
        "equipment_id": equipment_id,
        "equipment_name": equipment_name,
        "event_count": len(equipment_events),
        "failure_count": len(equipment_failures),
        "open_case_count": open_count,
        "critical_case_count": critical_count,
        "total_downtime_minutes": round(
            equipment_downtime,
            2,
        ),
        "average_repair_minutes": round(
            statistics.mean(
```

```
                    equipment_repair_values
                ),
                2,
            )
            if equipment_repair_values
            else 0.0,
            "mtbf_hours": mtbf_hours,
            "top_failure_categories": categories.most_common(3),
            "top_root_causes": root_causes.most_common(3),
        }
    )

equipment_metrics.sort(
    key=lambda item: (
        item["failure_count"],
        item["total_downtime_minutes"],
    ),
    reverse=True,
)

category_counts = Counter(
    event["failure_category"]
    for event in failure_events
    if event["failure_category"] != "Unknown"
)

root_cause_counts = Counter(
    event["root_cause"]
    for event in failure_events
    if event["root_cause"] != "Not recorded"
)

priority_counts = Counter(
    event["priority"]
    for event in real_events
)

status_counts = Counter(
    event["status"]
    for event in real_events
)

follow_up_count = sum(
    1
    for event in real_events
```

```
    if event["follow_up_required"]
)

safety_concern_count = sum(
    1
    for event in real_events
    if event["safety_concern"]
)

quality_concern_count = sum(
    1
    for event in real_events
    if event["quality_concern"]
)

production_stop_count = sum(
    1
    for event in real_events
    if event["production_stopped"]
)

return {
    "normalized_events": normalized,
    "total_events": len(real_events),
    "failure_events": len(failure_events),
    "closed_events": len(closed_events),
    "open_events": len(open_events),
    "critical_events": priority_counts.get(
        "Critical",
        0,
    ),
    "total_downtime_minutes": round(
        sum(downtime_values),
        2,
    ),
    "average_maintenance_time_invested_minutes": round(
    statistics.mean(maintenance_time_values),
    2,
)
    if maintenance_time_values
else None,

"median_maintenance_time_invested_minutes": round(
    statistics.median(maintenance_time_values),
    2,
```

```
    )
    if maintenance_time_values
    else None,

        "follow_up_required_count": follow_up_count,
        "safety_concern_count": safety_concern_count,
        "quality_concern_count": quality_concern_count,
        "production_stop_count": production_stop_count,
        "status_counts": dict(status_counts),
        "priority_counts": dict(priority_counts),
        "top_failure_categories": category_counts.most_common(10),
        "top_root_causes": root_cause_counts.most_common(10),
        "equipment_metrics": equipment_metrics,
    }

##########--------------------end of function that calculates reliability
metrics---------------######


#########----------------------Below code is for function that selects
relevant records-----------###########
def event_search_score(event: dict, question: str) -> int:
    """
    Score how relevant one maintenance event is to the user's question.
    """

    question_words = {
        word.lower()
        for word in question.replace("/", " ").replace("-", " ").split()
        if len(word) >= 3
    }

    searchable_text = " ".join(
        [
            event["event_id"],
            event["equipment_id"],
            event["equipment_name"],
            event["department"],
            event["location"],
            event["status"],
            event["priority"],
            event["failure_category"],
            event["failure_mode"],
            event["problem_description"],
            event["root_cause"],
```

```
            event["corrective_action"],
            event["parts_used"],
            event["follow_up_description"],
        ]
    ).lower()

    score = 0

    for word in question_words:
        if word in searchable_text:
            score += 2

    equipment_id = event["equipment_id"].lower()
    equipment_name = event["equipment_name"].lower()

    if equipment_id and equipment_id in question.lower():
        score += 20

    if (
        equipment_name
        and equipment_name != "unknown equipment"
        and equipment_name in question.lower()
    ):
        score += 20

    if event["priority"] == "Critical":
        score += 1

    if event["status"] in ACTIVE_STATUSES:
        score += 1

    return score


def select_relevant_events(
    normalized_events: list[dict],
    question: str,
    maximum_records: int = 30,
) -> list[dict]:
    """
    Select the most relevant events for the current AI question.
    """

    scored_events = [
        (
```

```
            event_search_score(event, question),
            event,
        )
        for event in normalized_events
        if event["status"] != "Draft"
    ]

    scored_events.sort(
        key=lambda item: (
            item[0],
            item[1]["reported_datetime"]
            or datetime.min.replace(tzinfo=timezone.utc),
        ),
        reverse=True,
    )

    relevant = [
        event
        for score, event in scored_events
        if score > 0
    ]

    if relevant:
        return relevant[:maximum_records]

    # When no keywords match, use the most recent records.
    recent_events = sorted(
        [
            event
            for event in normalized_events
            if event["status"] != "Draft"
        ],
        key=lambda event: (
            event["reported_datetime"]
            or datetime.min.replace(tzinfo=timezone.utc)
        ),
        reverse=True,
    )

    return recent_events[:maximum_records]

#########-------------------------End of code for function that selects
relevant records------------------##########
############--------------Below code is for function that prepares data
for the AI-------------------###############
```

```python
def datetime_for_ai(value):
    """
    Format datetimes for JSON and AI context.
    """
    if value is None:
        return None

    return value.isoformat()


def compact_event_for_ai(event: dict) -> dict:
    """
    Keep the useful event fields while limiting prompt size.
    """
    return {
        "event_id": event["event_id"],
        "equipment_id": event["equipment_id"],
        "equipment_name": event["equipment_name"],
        "department": event["department"],
        "location": event["location"],
        "status": event["status"],
        "priority": event["priority"],
        "event_type": event["event_type"],
        "failure_category": event["failure_category"],
        "failure_mode": event["failure_mode"],
        "problem_description": event["problem_description"],
        "root_cause": event["root_cause"],
        "corrective_action": event["corrective_action"],
        "parts_used": event["parts_used"],
        "follow_up_required": event["follow_up_required"],
        "follow_up_description": event["follow_up_description"],
        "production_stopped": event["production_stopped"],
        "safety_concern": event["safety_concern"],
        "quality_concern": event["quality_concern"],
        "reported_datetime": datetime_for_ai(
            event["reported_datetime"]
        ),
        "released_datetime": datetime_for_ai(
            event["released_datetime"]
        ),
       "maintenance_time_invested_minutes": event[
            "maintenance_time_invested_minutes"
        ],
        "downtime_minutes": event["downtime_minutes"],
```

```
    }


def build_ai_data_context(
    metrics: dict,
    relevant_events: list[dict],
) -> str:
    """
    Create a compact JSON package containing calculated reliability
    metrics and relevant maintenance records.
    """

    downtime_values = [
        float(field_value(event.get("DowntimeMinutes")) or 0)
        for event in relevant_events
        if field_value(event.get("DowntimeMinutes")) not in (None, "")
    ]

    summary = {
        "total_events": metrics["total_events"],
        "failure_events": metrics["failure_events"],
        "closed_events": metrics["closed_events"],
        "open_events": metrics["open_events"],
        "critical_events": metrics["critical_events"],
        "total_downtime_minutes": metrics["total_downtime_minutes"],

        "average_downtime_minutes": (
            round(statistics.mean(downtime_values), 2)
            if downtime_values
            else 0.0
        ),

        "average_maintenance_time_invested_minutes": metrics[
            "average_maintenance_time_invested_minutes"
        ],
        "median_maintenance_time_invested_minutes": metrics[
            "median_maintenance_time_invested_minutes"
        ],
        "follow_up_required_count": metrics[
            "follow_up_required_count"
        ],
        "safety_concern_count": metrics[
            "safety_concern_count"
        ],
        "quality_concern_count": metrics[
```

```
        "quality_concern_count"
    ],
    "production_stop_count": metrics[
        "production_stop_count"
    ],
    "status_counts": metrics["status_counts"],
    "priority_counts": metrics["priority_counts"],
    "top_failure_categories": metrics[
        "top_failure_categories"
    ],
    "top_root_causes": metrics[
        "top_root_causes"
    ],
    "equipment_metrics": metrics[
        "equipment_metrics"
    ][:20],
}

compact_events = []

for event in relevant_events[:50]:
    compact_events.append(
        {
            "event_id": field_value(event.get("EventID")),
            "equipment_id": field_value(event.get("EquipmentID")),
            "equipment_name": field_value(event.get("EquipmentName")),
            "department": field_value(event.get("Department")),
            "location": field_value(event.get("Location")),
            "status": field_value(event.get("Status")),
            "priority": field_value(event.get("Priority")),
            "reported_datetime_utc": field_value(
                event.get("ReportedDateTimeUTC")
            ),
            "assigned_datetime_utc": field_value(
                event.get("AssignedDateTimeUTC")
            ),
            "maintenance_start_utc": field_value(
                event.get("MaintenanceStartUTC")
            ),
            "repair_completed_utc": field_value(
                event.get("RepairCompletedUTC")
            ),
            "released_datetime_utc": field_value(
                event.get("ReleasedDateTimeUTC")
            ),
```

```
                "problem_description": field_value(
                    event.get("ProblemDescription")
                ),
                "failure_category": field_value(
                    event.get("FailureCategory")
                ),
                "failure_mode": field_value(event.get("FailureMode")),
                "root_cause": field_value(event.get("RootCause")),
                "corrective_action": field_value(
                    event.get("CorrectiveAction")
                ),
                "parts_used": field_value(event.get("PartsUsed")),
                "follow_up_required": field_value(
                    event.get("FollowUpRequired")
                ),
                "follow_up_description": field_value(
                    event.get("FollowUpDescription")
                ),
                "production_stopped": field_value(
                    event.get("ProductionStopped")
                ),
                "safety_concern": field_value(
                    event.get("SafetyConcern")
                ),
                "quality_concern": field_value(
                    event.get("QualityConcern")
                ),
                "response_minutes": field_value(
                    event.get("ResponseMinutes")
                ),
                "repair_minutes": field_value(
                    event.get("RepairMinutes")
                ),
                "waiting_release_minutes": field_value(
                    event.get("WaitingReleaseMinutes")
                ),
                "downtime_minutes": field_value(
                    event.get("DowntimeMinutes")
                ),
            }
        )

    return json.dumps(
        {
            "summary": summary,
```

```
            "maintenance_records": compact_events,
        },
        indent=2,
        default=str,
    )

    records = [
        compact_event_for_ai(event)
        for event in relevant_events
    ]

    payload = {
        "calculated_plant_summary": summary,
        "relevant_maintenance_records": records,
        "record_count_sent_to_ai": len(records),
    }

    return json.dumps(
        payload,
        ensure_ascii=False,
        indent=2,
        default=str,
    )

##########------------------function that prepares data for the AI_------
------------------##################

# temporary disable, to test drop down for equipment selection:
@st.cache_resource
#@st.cache_resource
def flow_client() -> FlowClient:
    return FlowClient(get_settings())


# OpenAI client function-----------------
@st.cache_resource
def openai_client() -> OpenAI:
    """Create and cache the OpenAI client."""

    try:
        api_key = st.secrets["openai"]["api_key"]
    except (KeyError, FileNotFoundError) as exc:
        raise RuntimeError(
            "The OpenAI API key is not configured in "
            ".streamlit/secrets.toml."
```

```
        ) from exc

    return OpenAI(api_key=api_key)
# ----------------------------------------


def settings():
    return get_settings()


def flows():
    return flow_client()


def utc_iso():
    return datetime.now(timezone.utc).isoformat().replace("+00:00", "Z")


def qp(name: str) -> str:
    value = st.query_params.get(name, "")
    return value[0] if isinstance(value, list) and value else str(value or
"")


def event_label(event: dict) -> str:
    return (
        f"{field_value(event.get('EventID', ''))} | "
        f"{field_value(event.get('EquipmentID', ''))} | "
        f"{field_value(event.get('Status', ''))}"
    )


def status_class(status: str) -> str:
    status = status.strip()
    if status == "Reported":
        return "status-reported"
    if status == "Assigned":
        return "status-assigned"
    if status == "Repair in Progress":
        return "status-progress"
    if status in {"Waiting for Parts", "Waiting for Contractor", "Awaiting
Operational Release"}:
        return "status-waiting"
    if status == "Closed - Returned to Operation":
        return "status-closed"
    return "status-default"
```

```
def render_status_badge(status: str) -> None:
    safe_status = status or "Unknown"
    st.markdown(
        f'<span class="status-badge {status_class(safe_status)}">●
{safe_status}</span>',
        unsafe_allow_html=True,
    )


def render_timeline(status: str) -> None:
    effective_status = status
    if status in {"Waiting for Parts", "Waiting for Contractor"}:
        effective_status = "Repair in Progress"

    try:
        current_index = STATUS_STEPS.index(effective_status)
    except ValueError:
        current_index = 0

    html = ['<div class="timeline">']
    for index, step in enumerate(STATUS_STEPS):
        css_class = ""
        if index < current_index:
            css_class = "done"
        elif index == current_index:
            css_class = "current"
        html.append(f'<div class="timeline-step
{css_class}">{step}</div>')
    html.append("</div>")
    st.markdown("".join(html), unsafe_allow_html=True)

####----------Gamification idea to display achievement----------------
#####

def render_repair_reward():
    reward = st.session_state.get("repair_reward")

    if not reward:
        return

    technician_name = escape(
        str(
            reward.get("technician")
```

```
        or "Maintenance Technician"
    ).strip()
)

equipment_name = escape(
    str(
        reward.get("equipment")
        or "Equipment"
    ).strip()
)

event_id = escape(
    str(
        reward.get("event_id")
        or "Maintenance Event"
    ).strip()
)

reward_css = """
<style>
.reward-card {
    background:
        radial-gradient(
            circle at top,
            rgba(42, 198, 255, 0.18),
            transparent 38%
        ),
        linear-gradient(
            145deg,
            #173e4d 0%,
            #102c38 55%,
            #091c26 100%
        );
    border: 2px solid #43d9ff;
    border-radius: 24px;
    padding: 36px 30px;
    margin: 18px 0 28px 0;
    text-align: center;
    color: #f4fbff;
    box-shadow:
        0 0 18px rgba(67, 217, 255, 0.45),
        0 0 50px rgba(67, 217, 255, 0.18);
    font-family: Arial, sans-serif;
}
```

```
.reward-icon {
    font-size: 58px;
    line-height: 1;
    margin-bottom: 12px;
}

.reward-label {
    color: #77e7ff;
    font-size: 14px;
    font-weight: 800;
    letter-spacing: 2px;
    margin-bottom: 10px;
}

.reward-title {
    color: #ffffff;
    font-size: 30px;
    font-weight: 900;
    margin-bottom: 14px;
}

.reward-subtitle {
    color: #ffd166;
    font-size: 22px;
    font-weight: 800;
    margin-bottom: 20px;
}

.reward-details {
    background: rgba(4, 21, 30, 0.78);
    border: 1px solid rgba(119, 231, 255, 0.25);
    border-radius: 16px;
    padding: 18px 20px;
    margin: 0 auto 22px auto;
    max-width: 720px;
    color: #dff7ff;
    font-size: 17px;
    line-height: 1.65;
}

.reward-details strong {
    color: #ffffff;
}

.xp-wrapper {
```

```
    max-width: 700px;
    margin: 0 auto 20px auto;
}

.xp-labels {
    display: flex;
    justify-content: space-between;
    margin-bottom: 8px;
    color: #dff7ff;
    font-size: 14px;
    font-weight: 700;
}

.xp-track {
    height: 16px;
    background: rgba(1, 13, 19, 0.9);
    border-radius: 999px;
    overflow: hidden;
    border: 1px solid rgba(119, 231, 255, 0.3);
}

.xp-fill {
    width: 100%;
    height: 100%;
    border-radius: 999px;
    background: linear-gradient(
        90deg,
        #10d7ff,
        #39f3b3,
        #ffd166
    );
    animation: rewardProgress 1.4s ease-out;
}

@keyframes rewardProgress {
    from {
        width: 0%;
    }

    to {
        width: 100%;
    }
}

.reward-points {
```

```
    display: inline-block;
    background: rgba(255, 209, 102, 0.13);
    border: 1px solid rgba(255, 209, 102, 0.55);
    color: #ffd166;
    border-radius: 999px;
    padding: 10px 22px;
    margin-bottom: 20px;
    font-size: 18px;
    font-weight: 900;
}

.reward-message {
    color: #cceef7;
    font-size: 15px;
    line-height: 1.8;
}
</style>
"""

reward_html = (
    '<div class="reward-card">'
    '<div class="reward-icon">⧖</div>'
    '<div class="reward-label">'
    'MAINTENANCE ACHIEVEMENT'
    '</div>'
    '<div class="reward-title">'
    'EQUIPMENT SUCCESSFULLY RESTORED'
    '</div>'
    '<div class="reward-subtitle">'
    f'Congratulations, {technician_name}!'
    '</div>'
    '<div class="reward-details">'
    f'<strong>{equipment_name}</strong> '
    'has been successfully repaired and submitted '
    'for operational release.'
    '<br><br>'
    'Maintenance Event:<br>'
    f'<strong>{event_id}</strong>'
    '</div>'
    '<div class="xp-wrapper">'
    '<div class="xp-labels">'
    '<span>Repair Completion</span>'
    '<span>100%</span>'
    '</div>'
    '<div class="xp-track">'
```

```
            '<div class="xp-fill"></div>'
            '</div>'
            '</div>'
            '<div class="reward-points">'
            '⭐ +100 Maintenance XP'
            '</div>'
            '<div class="reward-message">'
            '☑ Repair Completed<br>'
            '📷 Repair Evidence Submitted<br>'
            '🔧 Maintenance Record Updated<br>'
            '🚀 Operational Release Requested'
            '</div>'
            '</div>'
        )

        st.markdown(
            reward_css + reward_html,
            unsafe_allow_html=True,
        )

        st.balloons()

        if st.button(
            "Continue",
            key="close_repair_reward",
            use_container_width=True,
        ):
            st.session_state.pop("repair_reward", None)
            st.rerun()
#####--------------------end of gamification idea----------------------
########


def add_camera_photo(stage_key: str, camera_file) -> None:
    if camera_file is None:
        return
    key = f"camera_photos_{stage_key}"
    st.session_state.setdefault(key, [])
    raw = camera_file.getvalue()
    digest = hash(raw)
    if digest not in {item["digest"] for item in st.session_state[key]}:
        st.session_state[key].append(
            {
                "name": f"camera_{len(st.session_state[key]) + 1}.jpg",
                "bytes": raw,
```

```
                "digest": digest,
            }
        )


def camera_photos(stage_key: str):
    return [
        (item["name"], item["bytes"])
        for item in st.session_state.get(f"camera_photos_{stage_key}", [])
    ]


def clear_camera_photos(stage_key: str):
    st.session_state.pop(f"camera_photos_{stage_key}", None)


def collect_photos(uploaded_files, stage_key: str):
    photos = camera_photos(stage_key)
    photos.extend((uploaded.name, uploaded.getvalue()) for uploaded in
(uploaded_files or []))
    return photos


def validate_photos(photos):
    cfg = settings()
    if len(photos) > cfg.max_photos_per_stage:
        raise ValueError(f"Maximum {cfg.max_photos_per_stage} photos are
allowed.")
    max_bytes = cfg.max_image_mb * 1024 * 1024
    for name, content in photos:
        if len(content) > max_bytes:
            raise ValueError(f"{name} exceeds {cfg.max_image_mb} MB.")


def upload_photos(event_id, equipment_id, equipment_name, stage, photos,
uploaded_by):
    validate_photos(photos)
    results = []
    progress = st.progress(0, text="Preparing photographs...")
    for index, (original_name, raw) in enumerate(photos, 1):
        progress.progress(
            (index - 1) / len(photos),
            text=f"Uploading photo {index} of {len(photos)}...",
        )
        prepared = prepare_photo(
            raw,
```

```
            original_name,
            event_id,
            equipment_id,
            stage,
            index,
        )
        results.append(
            flows().upload_photo(
                {
                    "event_id": event_id,
                    "equipment_id": equipment_id,
                    "equipment_name": equipment_name,
                    "photo_stage": stage,
                    "photo_category": "Other",
                    "photo_caption": "",
                    "uploaded_by": uploaded_by,
                    "uploaded_datetime_utc": utc_iso(),
                    "is_primary_photo": index == 1,
                    "filename": prepared.filename,
                    "original_filename": prepared.original_filename,
                    "content_type": prepared.content_type,
                    "content_base64": prepared.content_base64,
                    "file_hash": prepared.sha256,
                    "size_bytes": prepared.size_bytes,
                }
            )
        )
    progress.progress(1.0, text=f"{len(results)} photos uploaded.")
    return results


def render_equipment(equipment):
    name = field_value(equipment.get("EquipmentName")) or "Equipment"
    equipment_id = field_value(equipment.get("EquipmentID"))
    location = field_value(equipment.get("Location")) or "Not specified"
    department = field_value(equipment.get("Department")) or "Not
specified"
    status = field_value(equipment.get("MachineStatus")) or "Unknown"

    st.markdown(
        f"""
        <div class="equipment-card">
            <div class="equipment-name">⚙ {name}</div>
            <div class="equipment-meta">
                ID: <strong>{equipment_id}</strong> ·
```

```
                Location: <strong>{location}</strong> ·
                Department: <strong>{department}</strong>
            </div>
        </div>
        """,
        unsafe_allow_html=True,
    )
    render_status_badge(status)


def dashboard_page():
    render_header(
        "Operations Overview",
        "Live maintenance workload and equipment-release status",
    )

    try:
        events = flows().query_events(include_closed=True)
    except FlowError as exc:
        st.error(f"Unable to load dashboard data: {exc}")
        return

    open_events = [
        event
        for event in events
        if field_value(event.get("Status")) in ACTIVE_STATUSES
        and field_value(event.get("Status")) != "Draft"
    ]
    critical = [
        event
        for event in open_events
        if field_value(event.get("Priority")) == "Critical"
    ]
    awaiting_release = [
        event
        for event in open_events
        if field_value(event.get("Status")) == "Awaiting Operational
Release"
    ]

    maintenance_times = []

    for event in events:
        invested_minutes = maintenance_time_invested_minutes(event)
```

```
    if invested_minutes is not None:
        maintenance_times.append(invested_minutes)

average_maintenance_time = (
    round(
        sum(maintenance_times) / len(maintenance_times),
        1,
    )
    if maintenance_times
    else None
)

columns = st.columns(4)
cards = [
    (
        "Open Repairs",
        len(open_events),
        "All active maintenance cases",
    ),
    (
        "Critical",
        len(critical),
        "Immediate-priority cases",
    ),
    (
        "Awaiting Release",
        len(awaiting_release),
        "Repairs pending verification",
    ),
    (
        "Maintenance Time Invested",
        format_duration(average_maintenance_time),
        "Average time from report submission to repair completion",
    ),
]
for column, (label, value, note) in zip(columns, cards):
    with column:
        st.markdown(
            f"""
            <div class="kpi-card">
                <div class="kpi-label">{label}</div>
                <div class="kpi-value">{value}</div>
                <div class="kpi-note">{note}</div>
            </div>
            """,
```

```
                unsafe_allow_html=True,
            )

    st.write("")
    left, right = st.columns([1.55, 1])

    with left:
        st.subheader("Active Maintenance Cases")
        if not open_events:
            st.success("No open maintenance cases.")
        else:
            for event in open_events[:12]:
                status = str(field_value(event.get("Status")) or "")
                name = field_value(event.get("EquipmentName")) or "Unnamed
equipment"
                equipment_id = field_value(event.get("EquipmentID")) or ""
                priority = field_value(event.get("Priority")) or "Not set"
                problem = field_value(event.get(" problemDescription")) or
"No description"
                st.markdown(
                    f"""
                    <div class="section-card">
                        <div style="display:flex;justify-content:space-
between;gap:1rem;align-items:flex-start">
                            <div>
                                <div style="font-
weight:850;color:#f3f7fa">{name} · {equipment_id}</div>
                                <div style="color:#a8bac6;font-
size:.84rem;margin-top:.25rem">
                                    Priority: <strong>{priority}</strong>
                                </div>
                                <div style="margin-
top:.55rem;color:#e4edf2">
                                    <strong>Problem:</strong> {problem}
                                </div>
                            </div>
                            <span class="status-badge
{status_class(status)}">● {status}</span>
                        </div>
                    </div>
                    """,
                    unsafe_allow_html=True,
                )

    with right:
```

```
        st.subheader("Workflow Status")
        status_counts = {}
        for event in open_events:
            status = str(field_value(event.get("Status")) or "Unknown")
            status_counts[status] = status_counts.get(status, 0) + 1

        if not status_counts:
            st.info("Workflow counts will appear when cases are active.")
        else:
            for status, count in sorted(status_counts.items()):
                st.markdown(
                    f"""
                    <div class="section-card" style="display:flex;align-
items:center;justify-content:space-between">
                        <span class="status-badge
{status_class(status)}">● {status}</span>
                        <strong style="font-size:1.25rem">{count}</strong>
                    </div>
                    """,
                    unsafe_allow_html=True,
                )

        st.subheader("Quick Actions")
        st.info(
            "Use the navigation menu to report a breakdown, work a case, "
            "release equipment, review history, or generate a QR code."
        )

def report_page():
    render_header(
        "Report Breakdown",
        "Create a traceable maintenance case with required before-repair
evidence",
    )

    # Check whether the user opened the page from an equipment QR code
    qr_equipment_id = qp("machine").strip()

    try:
        equipment_list = flows().get_equipment_master()

    except FlowError as exc:
        st.error(
            f"Unable to load equipment from EquipmentMaster: {exc}"
        )
```

```
        return

    if not equipment_list:
        st.warning(
            "No equipment records were found in the EquipmentMaster list."
        )
        return

    # Build dropdown labels showing Equipment ID and Equipment Name
    equipment_options = {}

    for eq in equipment_list:
        eq_id = str(
            field_value(eq.get("EquipmentID")) or ""
        ).strip()

        eq_name = str(
            field_value(eq.get("EquipmentName"))
            or "Unnamed Equipment"
        ).strip()

        if not eq_id:
            continue

        label = f"{eq_id} | {eq_name}"

        equipment_options[label] = eq


    if not equipment_options:
        st.warning(
            "No valid Equipment IDs were found in EquipmentMaster."
        )
        return


    option_labels = list(equipment_options.keys())


    # Default to the equipment from the QR code, when available
    default_index = 0

    if qr_equipment_id:
        qr_found = False
```

```
    for index, label in enumerate(option_labels):
        eq = equipment_options[label]

        eq_id = str(
            field_value(eq.get("EquipmentID")) or ""
        ).strip()

        if eq_id == qr_equipment_id:
            default_index = index
            qr_found = True
            break

    if not qr_found:
        st.warning(
            f"The equipment ID from the QR code "
            f"({qr_equipment_id}) was not found in EquipmentMaster."
        )


selected_label = st.selectbox(
    "Select Equipment",
    options=option_labels,
    index=default_index,
)


# Retrieve the complete selected equipment record
equipment = equipment_options[selected_label]

equipment_id = str(
    field_value(equipment.get("EquipmentID")) or ""
).strip()

equipment_name = str(
    field_value(equipment.get("EquipmentName"))
    or "Unnamed Equipment"
).strip()

render_equipment(equipment)

open_cases = [
    event
    for event in flows().query_events(
        equipment_id=equipment_id,
        include_closed=False,
```

```
        )
        if field_value(event.get("Status")) in ACTIVE_STATUSES
        and field_value(event.get("Status")) != "Draft"
    ]

    if open_cases:
        st.warning(
            "This machine already has an open case: "
            + event_label(open_cases[0])
        )

    st.subheader("Failure Information")
    with st.form("breakdown_form"):

#####------------Selection of Reporter--------------##########
        # Reporter list
        reporters = {
            "Select reporter": "",
            "Tomoatsu Maguchi": "tmaguchi@earthrise.com",
            "Rafael Mendez": "rmendez@earthrise.com",
            "Luis Andrade": "landrade@earthrise.com",
            "Esther Pasos": "epasos@earthrise.com",
        }

        reporter_name = st.selectbox(
            "Reported by",
            options=list(reporters.keys()),
            key="reporter_selection",
        )

        reporter_email = reporters.get(
            reporter_name,
            "",
        )

        if reporter_email:
            st.caption(
                f"Reporter email: {reporter_email}"
            )

#####------------Selection of Reporter--------------##########

        detail_1, detail_2 = st.columns(2)
        with detail_1:
            priority = st.selectbox(
```

```
                "Priority",
                ["Project", "Low", "Medium", "High", "Critical"],
                index=1,
            )
        with detail_2:
            failure_category = st.selectbox(
                "Failure category",
                [
                    "Mechanical",
                    "Electrical",
                    "Controls/Automation",
                    "Process",
                    "Leak",
                    "Safety Device",
                    "Unknown",
                    "Other",
                ],
            )

        description = st.text_area(
            "Describe the malfunction",
            height=140,
            placeholder="Describe what happened, abnormal sounds, alarms,
leakage, loss of function, and any immediate actions taken.",
        )

        checks = st.columns(3)
        with checks[0]:
            production_stopped = st.checkbox("Production stopped",
value=True)
        with checks[1]:
            safety_concern = st.checkbox("Safety concern")
        with checks[2]:
            quality_concern = st.checkbox("Product-quality concern")

        uploaded = st.file_uploader(
            "Select multiple before-repair photos",
            type=["jpg", "jpeg", "png", "webp"],
            accept_multiple_files=True,
            key="before_uploads",
        )
        submitted = st.form_submit_button(
            "🚨 Report Breakdown",
            type="primary",
            use_container_width=True,
```

```
        )

    with st.expander("📷 Take additional before-repair photographs"):
        camera = st.camera_input("Camera", key="before_camera")
        camera_actions = st.columns(2)
        with camera_actions[0]:
            if st.button(
                "Add camera photograph",
                key="add_before",
                use_container_width=True,
            ):
                add_camera_photo("before", camera)
                st.rerun()
        with camera_actions[1]:
            if st.button(
                "Clear camera photographs",
                key="clear_before",
                use_container_width=True,
            ):
                clear_camera_photos("before")
                st.rerun()

        count = len(camera_photos("before"))
        if count:
            st.success(f"{count} camera photograph(s) added.")

    if not submitted:
        return

    errors = []
    if not reporter_name.strip():
        errors.append("Reporter name is required.")
    if "@" not in reporter_email:
        errors.append("A valid reporter email is required.")
    if not description.strip():
        errors.append("Problem description is required.")

    photos = collect_photos(uploaded, "before")
    if not photos:
        errors.append("At least one before-repair photograph is
required.")

    if errors:
        for error in errors:
            st.error(error)
```

```
    return

event_id = (
    f"ME-{datetime.now(timezone.utc):%Y%m%d}-"
    f"{uuid4().hex[:8].upper()}"
)

try:
    flows().create_event(
        {
            "EventID": event_id,
            "EquipmentID": equipment_id,
            "EquipmentName": equipment.get("EquipmentName", ""),
            "Department": equipment.get("Department", ""),
            "Location": equipment.get("Location", ""),
            "EventType": "Breakdown",
            "CountsAsFailure": True,
            "Status": "Draft",
            "Priority": priority,
            "ReportedByName": reporter_name.strip(),
            "ReportedByEmail": reporter_email.strip().lower(),
            "ReportedDateTimeUTC": utc_iso(),
            "ProblemDescription": description.strip(),

"AuthenticatedUsername": current_username(),
"AuthenticatedDisplayName": current_display_name(),
"AuthenticatedRole": current_role(),

"FailureCategory": failure_category,
            "ProductionStopped": production_stopped,
            "SafetyConcern": safety_concern,
            "QualityConcern": quality_concern,

        }
    )

    photo_results = upload_photos(
        event_id,
        equipment_id,
        equipment.get("EquipmentName", ""),
        "Before Repair",
        photos,
        reporter_email.strip().lower(),
    )
```

```
        primary_url = ""
        if photo_results:
            first_photo_result = photo_results[0]
            photo_payload = first_photo_result.get(
                "photo",
                first_photo_result,
            )
            primary_url = field_value(
                photo_payload.get("web_url", "")
            )

        flows().update_event(
            event_id,
            "finalize_breakdown",
            {
                "status": "Reported",
                "before_photo_count": len(photo_results),
                "primary_before_photo_url": primary_url,
                "reported_by_email": reporter_email.strip().lower(),
            },
        )

        clear_camera_photos("before")
        st.success(f"Breakdown reported. Case number: {event_id}")
        st.info("Power Automate has sent or will send the breakdown
email.")

    except (FlowError, ValueError) as exc:
        st.error(f"Breakdown submission failed: {exc}")


def maintenance_page():
    render_header(
        "Maintenance Work Queue",
        "Assign, start, document, and complete repair work",
    )

#####-------Star of Technician Name Choices-----------------##########
    # Technician list
    technicians = {
        "Select technician": "",
        "Guillermo Garcia": "ggarcia@earthrise.com",
        "Israel Garcia": "igarcia@earthrise.com",
        "Pedro Lopez": "plopez@earthrise.com",
        "Pablo Mariano": "pmarino@earthrise.com",
```

```
        "Roberto Toji": "rtoji@earthrise.com",
    "Sergio Solano": "ssolano@earthrise.com"
    }

    technician_name = st.selectbox(
        "Select Technician",
        options=list(technicians.keys()),
        key="technician_selection",
    )

    technician_email = technicians.get(technician_name, "")

    if technician_email:
        st.caption(f"Technician email: {technician_email}")

#####-------End of Technician Name Choices------------------###########

    all_events = flows().query_events(include_closed=False)
    events = [
        event
        for event in all_events
        if field_value(event.get("Status")) in ACTIVE_STATUSES
        and field_value(event.get("Status")) != "Draft"
    ]

    if not events:
        st.success("No open maintenance cases.")
        return

    selected = st.selectbox(
        "Select maintenance case",
        events,
        format_func=event_label,
    )

    current_status = str(
        field_value(selected.get("Status", "")) or ""
    ).strip()

    assigned_name = str(
        field_value(selected.get("AssignedTechnicianName", "")) or ""
    ).strip()

    assigned_email = str(
        field_value(selected.get("AssignedTechnicianEmail", "")) or ""
```

```
    ).strip()

    name = field_value(selected.get("EquipmentName")) or "Equipment"
    equipment_id = field_value(selected.get("EquipmentID")) or ""
    priority = field_value(selected.get("Priority")) or "Not set"
    problem = field_value(selected.get("ProblemDescription")) or "No
description"

    st.markdown(
        f"""
        <div class="equipment-card">
            <div class="equipment-name">🔧 {name}</div>
            <div class="equipment-meta">
                Equipment ID: <strong>{equipment_id}</strong> ·
                Priority: <strong>{priority}</strong>
            </div>
            <div style="margin-top:.7rem;color:#e4edf2">{problem}</div>
        </div>
        """,
        unsafe_allow_html=True,
    )

    render_status_badge(current_status)
    render_timeline(current_status)

    if current_status == "Assigned":
        assigned_to = assigned_name or assigned_email or "another
technician"
        st.info(f"This case is already assigned to {assigned_to}.")
    elif current_status == "Repair in Progress":
        assigned_to = assigned_name or assigned_email or "a technician"
        st.info(f"Repair is already in progress by {assigned_to}.")

    before_url = field_value(selected.get("PrimaryBeforePhotoURL"))
    if before_url:
        st.link_button(
            "🖼 Open primary before photograph",
            before_url,
        )

    action_1, action_2 = st.columns(2)

    with action_1:
        assign_disabled = current_status != "Reported"
        if st.button(
```

```
            "👷 Accept / Assign Work",
            use_container_width=True,
            disabled=assign_disabled,
        ):
            if not technician_name.strip() or "@" not in technician_email:
                st.error("Enter technician name and email.")
            else:
                try:
                    flows().update_event(
                        selected["EventID"],
                        "assign",
                        {
                            "technician_name": technician_name.strip(),
                            "technician_email":
technician_email.strip().lower(),
                            "assigned_datetime_utc": utc_iso(),
                        },
                    )
                    st.success("Work was assigned successfully.")
                    st.rerun()
                except FlowError as exc:
                    st.error(f"Assignment failed: {exc}")

    with action_2:
        start_disabled = current_status != "Assigned"
        if st.button(
            "▶️ Start Repair",
            type="primary",
            use_container_width=True,
            disabled=start_disabled,
        ):
            if not technician_name.strip() or "@" not in technician_email:
                st.error("Enter technician name and email.")
            elif (
                assigned_email
                and technician_email.strip().lower() !=
assigned_email.lower()
            ):
                st.error("Only the assigned technician can start this
repair.")
            else:
                try:
                    flows().update_event(
                        selected["EventID"],
                        "start_repair",
```

```
                        {
                            "technician_name": technician_name.strip(),
                            "technician_email":
technician_email.strip().lower(),
                            "maintenance_start_utc": utc_iso(),
                        },
                    )
                    st.success("Repair was started successfully.")
                    st.rerun()
                except FlowError as exc:
                    st.error(f"Unable to start repair: {exc}")

    st.divider()
    st.subheader("Complete Repair")

    can_complete = current_status == "Repair in Progress"
    if not can_complete:
        st.info(
            "The repair form becomes active after the assigned technician
"
            "starts the repair."
        )

    with st.form("repair_form"):
        detail_1, detail_2 = st.columns(2)
        with detail_1:
            failure_mode = st.selectbox(
                "Failure mode",
                options=[
                    "",
                    "Preventive Maintenance",
                    "Mechanical Failure",
                    "Electrical Failure",
                    "Instrumentation / Sensor Failure",
                    "Control / PLC Failure",
                    "Motor Failure",
                    "Bearing Failure",
                    "Seal / Leakage Failure",
                    "Pump Failure",
                    "Valve Failure",
                    "Pneumatic Failure",
                    "Hydraulic Failure",
                    "Overheating",
                    "Blockage / Plugging",
                    "Wear / Deterioration",
```

```
                    "Operator / Human Error",
                    "Utility Failure",
                    "Other",
                ],
                index=0,
            )
    with detail_2:
        authorized_release_persons = {
            "Select authorized person": "",
            "Tomoatsu Maguchi — tmaguchi@earthrise.com":
"tmaguchi@earthrise.com",
            "Luis Andrade — landrade@earthrise.com":
"landrade@earthrise.com",
            "Esther Pasos — epasos@earthrise.com": "epasos@earthrise.com",
            "Jackie Montes — jmontes@earthrise.com":
"jmontes@earthrise.com",
            "Rebecca Deal — rdeal@earthrise.com": "rdeal@earthrise.com",
            "Rafa Mendez — rmendez@earthrise.com":
"rmendez@earthrise.com",
            "Nagatoshi Ide — nide@earthrise.com": "person3@earthrise.com",
        }

        selected_authorized_person = st.selectbox(
            "Person authorized to release machine",
            options=list(authorized_release_persons.keys()),
            disabled=not can_complete,
        )

        release_email =
authorized_release_persons[selected_authorized_person]

        root_cause = st.text_area(
            "Root cause",
            disabled=not can_complete,
        )
        corrective_action = st.text_area(
            "Corrective action",
            disabled=not can_complete,
        )
        parts_used = st.text_area(
            "Parts used",
            disabled=not can_complete,
        )
        follow_up_required = st.checkbox(
            "Follow-up required",
```

```
            disabled=not can_complete,
        )
        follow_up_description = st.text_area(
            "Follow-up description",
            disabled=not can_complete,
        )

        uploaded = st.file_uploader(
            "Select multiple after-repair photos",
            type=["jpg", "jpeg", "png", "webp"],
            accept_multiple_files=True,
            key="after_uploads",
            disabled=not can_complete,
        )

        completed = st.form_submit_button(
            "☑ Submit Repair for Operational Release",
            type="primary",
            disabled=not can_complete,
            use_container_width=True,
        )

    with st.expander("📷 Take additional after-repair photographs"):
        camera = st.camera_input(
            "Camera",
            key="after_camera",
            disabled=not can_complete,
        )

        photo_buttons = st.columns(2)
        with photo_buttons[0]:
            if st.button(
                "Add camera photograph",
                key="add_after",
                disabled=not can_complete,
                use_container_width=True,
            ):
                add_camera_photo("after", camera)
                st.rerun()
        with photo_buttons[1]:
            if st.button(
                "Clear camera photographs",
                key="clear_after",
                disabled=not can_complete,
                use_container_width=True,
```

```
            ):
                clear_camera_photos("after")
                st.rerun()

        count = len(camera_photos("after"))
        if count:
            st.success(f"{count} camera photograph(s) added.")

    if not completed:
        return

    errors = []

    if not technician_name.strip() or "@" not in technician_email:
        errors.append("Technician name and email are required.")

    if (
        assigned_email
        and technician_email.strip().lower() != assigned_email.lower()
    ):
        errors.append("Only the assigned technician can complete this
repair.")

    if not corrective_action.strip():
        errors.append("Corrective action is required.")

    if "@" not in release_email:
        errors.append("A valid release-authorized email is required.")

    photos = collect_photos(uploaded, "after")
    if not photos:
        errors.append("At least one after-repair photograph is required.")

    if errors:
        for error in errors:
            st.error(error)
        return

    try:
        photo_results = upload_photos(
            selected["EventID"],
            selected["EquipmentID"],
            field_value(selected.get("EquipmentName", "")),
            "After Repair",
            photos,
```

```
        technician_email.strip().lower(),
    )

    primary_url = ""
    if photo_results:
        first_photo_result = photo_results[0]
        photo_payload = first_photo_result.get(
            "photo",
            first_photo_result,
        )
        primary_url = field_value(
            photo_payload.get("web_url", "")
        )

    repair_values = {
    "technician_name": technician_name.strip(),
    "technician_email": technician_email.strip().lower(),
    "repair_completed_utc": utc_iso(),
    "failure_mode": failure_mode.strip(),
    "root_cause": root_cause.strip(),
    "corrective_action": corrective_action.strip(),
    "parts_used": parts_used.strip(),

    "authenticated_username": current_username(),
"authenticated_display_name": current_display_name(),
    "authenticated_role": current_role(),

    "follow_up_required": follow_up_required,
    "follow_up_description": follow_up_description.strip(),
    "after_photo_count": len(photo_results),
"primary_after_photo_url": primary_url,
    "release_authorized_email": release_email.strip().lower(),
"otp_expiration_minutes": settings().otp_expiration_minutes,
    "otp_max_attempts": settings().otp_max_attempts,
    "release_app_url": (
            f"{settings().app_base_url}/?event="
            f"{selected['EventID']}"
        ),
    }


    flows().update_event(
        selected["EventID"],
        "complete_repair",
        repair_values,
```

```
        )


        clear_camera_photos("after")

        st.session_state["repair_reward"] = {
            "technician": technician_name.strip(),
            "equipment": (
                field_value(selected.get("EquipmentName"))
                or field_value(selected.get("EquipmentID"))
                or "Equipment"
            ),
            "event_id": (
                field_value(selected.get("EventID"))
                or "Maintenance Event"
            ),
        }

        st.success(
            "Repair submitted successfully for operational release."
        )

        st.info(
            "Power Automate generated and emailed the one-time release
code."
        )

        render_repair_reward()

        st.stop()

    except (FlowError, ValueError) as exc:
        st.error(f"Repair completion failed: {exc}")


def release_page():
    if st.session_state.pop("release_success", False):
        st.success(
            "☑ OPERATIONAL RELEASE COMPLETED — "
            "The equipment has been successfully returned to operation."
        )
        st.balloons()

    render_header(
        "Operational Release",
```

```
        "Verify repair completion and return equipment safely to
operation",
    )

    awaiting = flows().query_events(
        status="Awaiting Operational Release",
        include_closed=False,
    )

    if not awaiting:
        st.success("No machines are awaiting operational release.")
        return

    event_from_url = qp("event")
    default_index = 0
    for index, event in enumerate(awaiting):
        if event.get("EventID") == event_from_url:
            default_index = index
            break

    selected = st.selectbox(
        "Select case",
        awaiting,
        index=default_index,
        format_func=event_label,
    )

    status = str(field_value(selected.get("Status")) or "")
    equipment_name = field_value(selected.get("EquipmentName")) or
"Equipment"
    equipment_id = field_value(selected.get("EquipmentID")) or ""
    corrective_action = (
        field_value(selected.get("CorrectiveAction"))
        or "No corrective action recorded."
    )

    st.markdown(
        f"""
        <div class="equipment-card">
            <div class="equipment-name">☑ {equipment_name}</div>
            <div class="equipment-meta">
                Equipment ID: <strong>{equipment_id}</strong>
            </div>
            <div style="margin-top:.7rem;color:#e4edf2">
                <strong>Corrective action:</strong> {corrective_action}
```

```
            </div>
        </div>
        """,
        unsafe_allow_html=True,
    )

    render_timeline(status)

    after_url = field_value(selected.get("PrimaryAfterPhotoURL"))
    if after_url:
        st.link_button(
            "🖼 Open primary after photograph",
            after_url,
        )

    identity_1, identity_2 = st.columns(2)
    with identity_1:
        name = st.text_input("Your name")
    with identity_2:
        email = st.text_input("Your email")

    code = st.text_input(
        "Six-digit release code",
        max_chars=6,
        placeholder="000000",
    )
    confirmed = st.checkbox(
        "I verified that the equipment is safe, functional, and ready for
operation."
    )

    if st.button(
        "🔓 Release Machine to Operation",
        type="primary",
        use_container_width=True,
    ):
        if not name.strip() or "@" not in email:
            st.error("Enter your name and a valid email.")
            return
        if len(code.strip()) != 6 or not code.strip().isdigit():
            st.error("Enter the six-digit release code.")
            return
        if not confirmed:
            st.error("You must confirm operational verification.")
            return
```

```
        try:
            result = flows().release_event(
                selected["EventID"],
                name.strip(),
                email.strip().lower(),
                code.strip(),
                confirmed,
            )

            st.session_state["release_success"] = True
            st.session_state["release_downtime"] =
result.get("DowntimeMinutes", 0)

            st.rerun()

        except FlowError as exc:
            st.error(str(exc))


def history_page():
    render_header(
        "Equipment History",
        "Review completed and active maintenance records by equipment",
    )

    # ------------------------------------------------------------
    # Load EquipmentMaster for dropdown selection
    # ------------------------------------------------------------
    try:
        equipment_list = flows().get_equipment_master()

    except FlowError as exc:
        st.error(
            f"Unable to load equipment from EquipmentMaster: {exc}"
        )
        return

    if not equipment_list:
        st.warning(
            "No equipment records were found in the EquipmentMaster list."
        )
        return

    # ------------------------------------------------------------
```

```
# Build dropdown labels:
# EquipmentID | EquipmentName
# ------------------------------------------------------------
equipment_options = {}

for eq in equipment_list:
    eq_id = str(
        field_value(eq.get("EquipmentID")) or ""
    ).strip()

    eq_name = str(
        field_value(eq.get("EquipmentName"))
        or "Unnamed Equipment"
    ).strip()

    if not eq_id:
        continue

    label = f"{eq_id} | {eq_name}"
    equipment_options[label] = eq

if not equipment_options:
    st.warning(
        "No valid Equipment IDs were found in EquipmentMaster."
    )
    return

option_labels = list(equipment_options.keys())

# ------------------------------------------------------------
# If History page was opened from a QR link,
# automatically select that equipment when possible.
# ------------------------------------------------------------
qr_equipment_id = qp("machine").strip()

default_index = 0

if qr_equipment_id:
    for index, label in enumerate(option_labels):
        eq = equipment_options[label]

        eq_id = str(
            field_value(eq.get("EquipmentID")) or ""
        ).strip()
```

```
            if eq_id == qr_equipment_id:
                default_index = index
                break

    # ------------------------------------------------------------
    # Equipment dropdown
    # ------------------------------------------------------------
    selected_label = st.selectbox(
        "Select Equipment",
        options=option_labels,
        index=default_index,
        key="history_equipment_selector",
    )

    selected_equipment = equipment_options[selected_label]

    equipment_id = str(
        field_value(
            selected_equipment.get("EquipmentID")
        ) or ""
    ).strip()

    if not equipment_id:
        st.warning("The selected equipment does not have an Equipment
ID.")
        return

    # Optional equipment information card
    render_equipment(selected_equipment)

    # ------------------------------------------------------------
    # Retrieve maintenance history for selected equipment
    # ------------------------------------------------------------
    try:
        events = flows().query_events(
            equipment_id=equipment_id,
            include_closed=True,
        )

    except FlowError as exc:
        st.error(
            f"Unable to load maintenance history: {exc}"
        )
        return
```

```
if not events:
    st.info(
        f"No maintenance history was found for {equipment_id}."
    )
    return

# ------------------------------------------------------------
# Calculate history KPIs
# ------------------------------------------------------------
closed = [
    event
    for event in events
    if field_value(event.get("Status"))
    == "Closed - Returned to Operation"
]

downtime_values = [
    float(
        field_value(
            event.get("DowntimeMinutes")
        ) or 0
    )
    for event in closed
    if field_value(
        event.get("DowntimeMinutes")
    ) not in (None, "")
]

maintenance_times = [
    maintenance_time_invested_minutes(event)
    for event in events
]

maintenance_times = [
    value
    for value in maintenance_times
    if value is not None
]

average_invested_time = (
    round(
        sum(maintenance_times)
        / len(maintenance_times),
        1,
    )
```

```
    if maintenance_times
    else None
)

metrics = st.columns(4)

metrics[0].metric(
    "Total Cases",
    len(events),
)

metrics[1].metric(
    "Closed Cases",
    len(closed),
)

metrics[2].metric(
    "Average Downtime",
    (
        f"{round(sum(downtime_values) / len(downtime_values), 1)} min"
        if downtime_values
        else "Not available"
    ),
)

metrics[3].metric(
    "Maintenance Time Invested",
    format_duration(
        average_invested_time
    ),
)

# ------------------------------------------------------------
# Maintenance event table
# ------------------------------------------------------------
rows = [
    {
        "Event ID": event.get("EventID"),
        "Status": field_value(
            event.get("Status")
        ),
        "Priority": field_value(
            event.get("Priority")
        ),
        "Reported UTC": event.get(
```

```
                "ReportedDateTimeUTC"
            ),
            "Released UTC": event.get(
                "ReleasedDateTimeUTC"
            ),
            "Downtime min": event.get(
                "DowntimeMinutes"
            ),
            "Maintenance Time Invested": format_duration(
                maintenance_time_invested_minutes(
                    event
                )
            ),
            "Before photos": event.get(
                "BeforePhotoCount"
            ),
            "After photos": event.get(
                "AfterPhotoCount"
            ),
            "Problem": event.get(
                "ProblemDescription"
            ),
            "Corrective action": event.get(
                "CorrectiveAction"
            ),
        }
        for event in events
    ]

    st.dataframe(
        rows,
        use_container_width=True,
        hide_index=True,
    )

def qr_page():
    render_header(
        "Generate QR Code",
        "Create a durable equipment link for rapid breakdown reporting",
    )

    # ------------------------------------------------------------
    # Load EquipmentMaster for dropdown selection
    # ------------------------------------------------------------
    try:
```

```
    equipment_list = flows().get_equipment_master()

except FlowError as exc:
    st.error(
        f"Unable to load equipment from EquipmentMaster: {exc}"
    )
    return

if not equipment_list:
    st.warning(
        "No equipment records were found in the EquipmentMaster list."
    )
    return

# ------------------------------------------------------------
# Build dropdown labels:
# EquipmentID | EquipmentName
# ------------------------------------------------------------
equipment_options = {}

for eq in equipment_list:
    eq_id = str(
        field_value(eq.get("EquipmentID")) or ""
    ).strip()

    eq_name = str(
        field_value(eq.get("EquipmentName"))
        or "Unnamed Equipment"
    ).strip()

    if not eq_id:
        continue

    label = f"{eq_id} | {eq_name}"
    equipment_options[label] = eq

if not equipment_options:
    st.warning(
        "No valid Equipment IDs were found in EquipmentMaster."
    )
    return

option_labels = list(equipment_options.keys())

# ------------------------------------------------------------
```

```
# Equipment dropdown
# ------------------------------------------------------------
selected_label = st.selectbox(
    "Select Equipment",
    options=option_labels,
    key="qr_equipment_selector",
)

equipment = equipment_options[selected_label]

equipment_id = str(
    field_value(
        equipment.get("EquipmentID")
    ) or ""
).strip()

equipment_name = str(
    field_value(
        equipment.get("EquipmentName")
    )
    or "Equipment"
).strip()

location = str(
    field_value(
        equipment.get("Location")
    )
    or "Not specified"
).strip()

department = str(
    field_value(
        equipment.get("Department")
    )
    or "Not specified"
).strip()

if not equipment_id:
    st.error(
        "The selected equipment does not have a valid Equipment ID."
    )
    return

# ------------------------------------------------------------
# Generate durable QR link
```

```python
# ------------------------------------------------------------
url = (
    f"{settings().app_base_url}"
    f"/?machine={equipment_id}"
)

image = qrcode.make(url)

output = BytesIO()

image.save(
    output,
    format="PNG",
)

# ------------------------------------------------------------
# Display QR code and equipment information
# ------------------------------------------------------------
left, right = st.columns(
    [1, 1.4]
)

with left:
    st.image(
        output.getvalue(),
        width=340,
    )

with right:
    st.subheader(
        equipment_name
    )

    st.write(
        f"**Equipment ID:** {equipment_id}"
    )

    st.write(
        f"**Location:** {location}"
    )

    st.write(
        f"**Department:** {department}"
    )
```

```
        st.markdown(
            "**QR destination:**"
        )

        st.code(url)

        st.download_button(
            "⬇️ Download QR Code",
            output.getvalue(),
            file_name=(
                f"{equipment_id}_maintenance_qr.png"
            ),
            mime="image/png",
            use_container_width=True,
        )

# ----------------------AI assistent page start--------------------------
---
def ai_assistant_page():
    render_header(
        "Plant Reliability AI",
        "Maintenance-history analysis, reliability insights, and
preventive-action guidance",
    )

    st.warning(
        "AI recommendations are advisory only. Follow approved procedures,
"
        "equipment manuals, lockout/tagout requirements, and plant safety
"
        "rules. Operational release remains the responsibility of
authorized "
        "plant personnel."
    )

    # ---------------------------------------
    # Retrieve maintenance records
    # ---------------------------------------
    try:
        with st.spinner(
            "Loading maintenance records from Power Automate and
SharePoint..."
        ):
            raw_events = flows().query_events(
                include_closed=True
```

```
        )

except FlowError as exc:
    st.error(
        "The AI could not retrieve maintenance records. "
        f"Power Automate returned: {exc}"
    )
    return

if not raw_events:
    st.info(
        "No maintenance records were returned. Confirm that Flow 2 "
        "returns active and closed maintenance records."
    )
    return

# ----------------------------------------
# Calculate plant reliability metrics
# ----------------------------------------
try:
    metrics = calculate_reliability_metrics(
        raw_events
    )
except Exception as exc:
    st.error(
        "The maintenance records were loaded, but the reliability "
        f"calculations failed: {exc}"
    )
    return

st.success(
    f"Loaded {metrics['total_events']} maintenance records. "
    f"Detected {metrics['failure_events']} breakdown/failure records."
)

# ----------------------------------------
# Display calculated KPI cards
# ----------------------------------------
kpi_1, kpi_2, kpi_3, kpi_4 = st.columns(4)

kpi_1.metric(
    "Maintenance Records",
    metrics["total_events"],
)
```

```
    kpi_2.metric(
        "Failure Events",
        metrics["failure_events"],
    )

    kpi_3.metric(
        "Maintenance Time Invested",
        format_duration(
            metrics[
                "average_maintenance_time_invested_minutes"
            ]
        ),
    )

    kpi_4.metric(
        "Total Downtime",
        f"{metrics['total_downtime_minutes'] / 60:.1f} hr",
    )

    st.caption(
        "Maintenance Time Invested represents the total elapsed time from
"
        "breakdown reporting until repair completion. "
        "Estimated Time Between Failures is based on the calendar time
between "
        "recorded failures and should be considered an estimate until
machine "
        "runtime-hour data becomes available."
    )

    # ----------------------------------------
    # Equipment ranking
    # ----------------------------------------
    st.subheader("Equipment Reliability Ranking")

    equipment_table = []

    for item in metrics["equipment_metrics"][:20]:
        equipment_table.append(
            {
                "Equipment ID": item["equipment_id"],
                "Equipment": item["equipment_name"],
                "Failures": item["failure_count"],
                "Open Cases": item["open_case_count"],
                "Critical Cases": item[
```

```
                "critical_case_count"
            ],
            "Downtime Hours": round(
                item["total_downtime_minutes"] / 60,
                2,
            ),
            "Average Repair Min": item[
                "average_repair_minutes"
            ],
            "Calendar MTBF Hours": (
                item["mtbf_hours"]
                if item["mtbf_hours"] is not None
                else ""
            ),
        }
    )

if equipment_table:
    st.dataframe(
        equipment_table,
        use_container_width=True,
        hide_index=True,
    )
else:
    st.info(
        "There are not enough maintenance records to create "
        "an equipment ranking."
    )

# ----------------------------------------
# Filters for the AI
# ----------------------------------------
st.subheader("Ask the Reliability AI")

filter_1, filter_2 = st.columns(2)

equipment_options = [
    "All equipment"
] + sorted(
    {
        (
            f"{item['equipment_id']} | "
            f"{item['equipment_name']}"
        )
        for item in metrics["equipment_metrics"]
```

```
    }
)

with filter_1:
    selected_equipment = st.selectbox(
        "Optional equipment filter",
        equipment_options,
    )

with filter_2:
    maximum_records = st.selectbox(
        "Maximum records sent to AI",
        [10, 20, 30, 40, 50],
        index=2,
        help=(
            "A larger number provides more history but may increase "
            "API cost and response time."
        ),
    )

analysis_type = st.selectbox(
    "Analysis focus",
    [
        "General reliability analysis",
        "Recurring failures",
        "Downtime and MTTR",
        "Preventive maintenance priorities",
        "Root-cause trends",
        "Spare-parts recommendations",
        "Safety and quality concerns",
        "Open-work-order priorities",
    ],
)

if "ai_messages" not in st.session_state:
    st.session_state["ai_messages"] = []

for message in st.session_state[
    "ai_messages"
]:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

prompt = st.chat_input(
```

```
        "Example: Which equipment should receive preventive maintenance
first, and why?"
    )

    clear_column, refresh_column = st.columns(2)

    with clear_column:
        if st.button(
            "🗑 Clear AI Conversation",
            use_container_width=True,
        ):
            st.session_state["ai_messages"] = []
            st.rerun()

    with refresh_column:
        if st.button(
            "🔄 Refresh Maintenance Data",
            use_container_width=True,
        ):
            st.cache_data.clear()
            st.rerun()

    if not prompt:
        return

    st.session_state["ai_messages"].append(
        {
            "role": "user",
            "content": prompt,
        }
    )

    with st.chat_message("user"):
        st.markdown(prompt)

    # ----------------------------------------
    # Select relevant maintenance records
    # ----------------------------------------
    search_question = (
        f"{analysis_type}. {prompt}"
    )

    relevant_events = select_relevant_events(
        metrics["normalized_events"],
        search_question,
```

```
        maximum_records=maximum_records,
    )

    if selected_equipment != "All equipment":
        selected_equipment_id = (
            selected_equipment.split("|", 1)[0].strip()
        )

        equipment_events = [
            event
            for event in metrics["normalized_events"]
            if event["equipment_id"]
            == selected_equipment_id
        ]

        equipment_events.sort(
            key=lambda event: (
                event["reported_datetime"]
                or datetime.min.replace(
                    tzinfo=timezone.utc
                )
            ),
            reverse=True,
        )

        relevant_events = equipment_events[
            :maximum_records
        ]

    ai_data_context = build_ai_data_context(
        metrics,
        relevant_events,
    )

    # ----------------------------------------
    # Reliability-engineer instructions
    # ----------------------------------------
    system_instructions = """
You are the Plant Reliability AI for an industrial manufacturing facility.

You are given calculated maintenance metrics and selected maintenance
records
retrieved from the plant's maintenance system.

Your responsibilities:
```

1. Identify equipment with the greatest reliability risk.
2. Detect repeated failure categories, failure modes, and root causes.
3. Explain downtime and repair-time trends.
4. Recommend preventive and predictive maintenance priorities.
5. Recommend inspections and spare-parts planning based only on the records.
6. Identify missing or poor-quality maintenance documentation.
7. Separate confirmed findings from hypotheses.
8. Cite the Event IDs supporting important findings.
9. State the number of records used in the analysis.
10. Clearly state when the available data is insufficient.

Calculation rules:

- Use the calculated metrics supplied by the application.
- Do not recalculate or invent totals when a calculated value is provided.
- 'Calendar MTBF Hours' is elapsed calendar time between recorded failures.
- Do not call calendar MTBF operating-hour MTBF.
- True operating-hour MTBF requires runtime or production-hour data.
- Do not interpret blank values as zero unless the calculated summary does so.
- Do not claim causation from correlation alone.
- Do not invent equipment manuals, specifications, readings, or procedures.

Response structure:

Executive Summary
Provide a brief summary of the most important findings.

Evidence From Maintenance Records
Provide numbers, equipment names, and supporting Event IDs.

Reliability Risks
Rank the most important risks.

Recommended Actions
Separate recommendations into:
- Immediate
- Within 30 days
- Longer-term

Data Quality Limitations

```
Identify missing root causes, repair times, downtime values, or
inconsistent
records that reduce confidence.

Safety requirements:

- Never recommend bypassing guards, interlocks, alarms, emergency stops,
or
  safety devices.
- Never advise intrusive maintenance without appropriate isolation and
  lockout/tagout.
- Operational release must remain with authorized plant personnel.

Metric definition:

"Maintenance Time Invested" is the elapsed calendar time between
ReportedDateTimeUTC and RepairCompletedUTC.

It begins when the breakdown report is submitted and ends when the
technician submits the completed repair.

It is not MTTR and must never be described as MTTR.

It may include assignment delay, technician response delay, active repair,
waiting for parts, waiting for contractors, and other elapsed time before
repair completion.

"""

    previous_conversation = ""

    for message in st.session_state[
        "ai_messages"
    ][-8:]:
        previous_conversation += (
            f"\n{message['role'].upper()}: "
            f"{message['content']}\n"
        )

    user_input = f"""
Requested analysis focus:
{analysis_type}

Optional equipment filter:
{selected_equipment}
```

```
Maintenance data:
{ai_data_context}

Recent conversation:
{previous_conversation}

Current question:
{prompt}
"""

    # ----------------------------------------
    # Send data and question to OpenAI
    # ----------------------------------------
    try:
        model = st.secrets["openai"].get(
            "model",
            "gpt-4.1-mini",
        )

        with st.chat_message("assistant"):
            with st.spinner(
                "Analyzing plant maintenance history..."
            ):
                response = (
                    openai_client()
                    .responses.create(
                        model=model,
                        instructions=system_instructions,
                        input=user_input,
                    )
                )

                answer = response.output_text

            st.markdown(answer)

            st.caption(
                f"Analysis used {len(relevant_events)} selected "
                f"maintenance records from {metrics['total_events']} "
                "available records."
            )

        st.session_state["ai_messages"].append(
            {
```

```
                "role": "assistant",
                "content": answer,
            }
        )

    except Exception as exc:
        st.error(
            "The reliability AI request failed. Confirm the API key, "
            "working model name, API billing, and Streamlit Secrets. "
            f"Technical details: {exc}"
        )

# ----------------------AI assistent page end----------------------------
-

def render_add_equipment_page():
    render_header(
        "Add New Equipment",
        "Register equipment in the Equipment Master List",
    )

    st.warning(
        """
        ⚠ **Equipment Registration Control**

        All new equipment must receive **prior approval from the Plant
Manager**
        before being entered into the Equipment Master List.

        **EET must be notified** of every new equipment registration.
        """
    )

    st.markdown("### Equipment Information")

    col1, col2 = st.columns(2)

    with col1:
        equipment_id = st.text_input(
            "Equipment ID *",
            placeholder="Example: P-001",
        )

        equipment_name = st.text_input(
            "Equipment Name *",
```

```
        placeholder="Example: Transfer Pump No. 1",
    )

    department = st.selectbox(
        "Department *",
        [
            "",
            "Production",
            "Maintenance",
            "Quality",
            "Warehouse",
            "Utilities",
            "Engineering",
            "Other",
        ],
    )

    criticality = st.selectbox(
      "Criticality *",
      [
          "",
          "Low",
          "Medium",
          "High",
          "Critical",
      ],
  )

with col2:
    location = st.text_input(
        "Location *",
        placeholder="Example: LB-Dry",
    )

    equipment_type = st.text_input(
        "Equipment Type",
        placeholder="Example: Pump",
    )

    manufacturer = st.text_input(
        "Manufacturer",
    )

model = st.text_input("Model")
serial_number = st.text_input("Serial Number")
```

```
    st.markdown("### Authorization")

    plant_manager_approval = st.checkbox(
        "I confirm that prior approval from the Plant Manager has been
obtained."
    )

    eet_notified = st.checkbox(
        "I confirm that EET has been notified of this equipment
registration."
    )

    added_by = st.text_input(
        "Name of person entering equipment *"
    )

    notes = st.text_area(
        "Additional notes"
    )

    can_submit = (
        bool(equipment_id.strip())
        and bool(equipment_name.strip())
        and bool(department)
        and bool(criticality)
        and bool(location.strip())
        and bool(added_by.strip())
        and plant_manager_approval
        and eet_notified
    )

    if st.button(
        "Add Equipment to Master List",
        type="primary",
        disabled=not can_submit,
        use_container_width=True,
    ):
        try:
            result = flows().add_equipment(
                {
                    "EquipmentID": equipment_id.strip(),
                    "EquipmentName": equipment_name.strip(),
                    "Department": department,
                    "Criticality": criticality,
```

```
                    "Location": location.strip(),
                    "EquipmentType": equipment_type.strip(),
                    "Manufacturer": manufacturer.strip(),
                    "Model": model.strip(),
                    "SerialNumber": serial_number.strip(),
                    "PlantManagerApproved": plant_manager_approval,
                    "AddedByName": added_by.strip(),
                    "RegistrationNotes": notes.strip(),
                }
            )

            st.success(
                f"Equipment {equipment_id} - {equipment_name} "
                "was successfully added to EquipmentMaster."
            )

        except FlowError as exc:
            st.error(
                f"Equipment registration failed: {exc}"
            )

def main():
    if not check_password():
        st.stop()

    apply_theme()

    st.sidebar.markdown("## 🛠️ Maintenance")
    st.sidebar.caption("Equipment Reliability System")

    role = current_role()

    display_name = (
        current_display_name()
        or current_username()
        or "Authorized User"
    )

    st.sidebar.markdown(
        f"**👤 {display_name}**"
    )

    st.sidebar.caption(
        f"Access level: {role.title()}"
    )
```

```
    if st.sidebar.button(
        "🚪 Log Out",
        use_container_width=True,
    ):
        keys_to_clear = [
            "authenticated",
            "username",
            "display_name",
            "role",
            "login_username",
            "login_password",
        ]

        for key in keys_to_clear:
            st.session_state.pop(key, None)

        st.rerun()

    st.sidebar.divider()

    allowed_pages = ROLE_PAGES.get(
        role,
        ROLE_PAGES["viewer"],
    )

    page = st.sidebar.radio(
        "Navigation",
        allowed_pages,
        label_visibility="collapsed",
    )


    st.sidebar.divider()
    st.sidebar.caption(
        "QR reporting · Photo evidence · One-time release code · Power BI
metrics system 100% developed by EEC"

    )


    st.sidebar.caption(
    f"Equipment Maintenance Management System · v{APP_VERSION}"
    )
```

```
    try:
        {
            "Dashboard": dashboard_page,
            "Report Breakdown": report_page,
            "Maintenance Queue": maintenance_page,
            "Operational Release": release_page,
            "Equipment History": history_page,
            "Add Equipment": render_add_equipment_page,
            "Generate QR Code": qr_page,
            "AI Assistant": ai_assistant_page,
        }[page]()

    except RuntimeError as exc:
        st.error(str(exc))

    except FlowError as exc:
        st.error(f"Power Automate error: {exc}")


if __name__ == "__main__":
    main()
```