INFORME: DASHBOARD SATÈL·LIT CANSAT (DIRECTOR DE VOL)

Dashboard en temps real per monitoritzar el satèl·lit SatPi26 i simular un centre de control. Desenvolupat amb Python i Streamlit per analitzar i visualitzar dades.

1. Importació de llibreries

El codi comença amb la importació de diverses llibreries necessàries:

- **os, sys, subprocess, webbrowser:** per gestionar l'execució del programa i obrir el dashboard automàticament
- **time:** per controlar l'actualització en temps real
- **numpy i pandas:** per generar i gestionar dades
- **plotly:** per crear gràfics interactius
- **streamlit:** per construir la interfície visual

```python
# --- IMPORTACIÓ DE LLIBRERIES ---
import os              # Per interactuar amb el sistema operatiu (fitxers, rutes)
import sys             # Per gestionar paràmetres i funcions del sistema Python
import subprocess      # Per executar el servidor Streamlit des de dins del codi
import time            # Per controlar els temps d'espera i refresc
import webbrowser      # Per obrir automàticament el navegador amb el dashboard
import numpy as np     # Per càlculs matemàtics i generació de soroll aleatori
import pandas as pd    # Per gestionar l'estructura de les dades (DataFrames)
import plotly.graph_objects as go  # Per crear gràfics complexos (agulles, indicadors)
import plotly.express as px        # Per crear gràfics ràpids (mapes, línies, 3D)
```

2. Llançador automàtic del dashboard

El programa inclou un sistema que permet executar automàticament el dashboard amb Streamlit:

- Comprova si Streamlit està actiu
- Si no ho està, inicia un servidor al port **8505**
- Obre automàticament el navegador

```python
# --- 1. LLANÇADOR BLINDAT (Permet executar des de Thonny directament) ---
if __name__ == "__main__":                    # Si aquest fitxer és el principal:
    if "streamlit" not in sys.modules:        # Si Streamlit no s'està executant:
        ruta_real = os.path.abspath(__file__)   # Obté la ruta d'aquest fitxer .py
        try:
            # Executa el servidor Streamlit en un procés separat (Port 8505)
            subprocess.Popen([sys.executable, "-m", "streamlit", "run", ruta_real, "--server.headless", "true", "--server.port", "8505"])
            time.sleep(4)                     # Espera 4 segons a que el servidor aixequi
            webbrowser.open("http://localhost:8505") # Obre el navegador a l'adreça local
        except Exception as e:
            print(f"Error: {e}")              # Mostra l'error si el llançament falla
        sys.exit()                            # Tanca el procés inicial de Python
```

Això permet executar el programa directament des de Thonny o qualsevol entorn sense passos manuals. *línia 21 a la 33*

3. Configuració visual i estil (CSS)

S'utilitza CSS personalitzat per donar un aspecte futurista:

- Colors foscos amb tons neó
- Tipografies tecnològiques (Orbitron i Share Tech Mono)
- Targetes de dades (metric-card)
- Indicadors visuals de fases

Aquesta part és important perquè millora la **usabilitat i l'estètica** del dashboard.

línia 36 a la 70, aix

```python
# --- 2. CONFIGURACIÓ VISUAL I ESTIL (CSS) ---
# Defineix el títol de la pestanya i l'amplada de la pàgina
st.set_page_config(page_title="SatPi2026 DIRECTOR DE VOL", layout="wide", initial_sidebar_state="expanded")

# Injecció de codi CSS per personalitzar l'aparença (Colors de neó, tipografies)
st.markdown("""
    <style>
    @import url('https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&family=Share+Tech+Mono&display=swap');

    /* Estil de fons i tipografia general */
    .stApp { background-color: #05070a; font-family: 'Share Tech Mono', monospace; color: #e0e0e0; }
    h1, h2, h3, h4 { font-family: 'Orbitron', sans-serif; color: #00e6e6 !important; text-shadow: 0 0 10px rgba(0, 230, 230, 0.3); }

    /* Disseny de les caixes de dades (Targetes) */
    .metric-card {
        background: rgba(10, 15, 25, 0.9);
        border: 1px solid #005f5f;
        border-radius: 8px;
        padding: 15px;
        text-align: center;
        box-shadow: inset 0 0 10px rgba(0, 230, 230, 0.1);
        margin-bottom: 10px;
    }
    .metric-title { color: #8b949e; font-size: 0.8rem; text-transform: uppercase; margin-bottom: 5px; }
    .metric-value { color: #00e6e6; font-size: 1.8rem; font-weight: bold; }

    /* Estil per les fases de missió (Activa vs Inactiva) */
    .phase-active { background-color: #00e6e6; color: #000; font-weight: bold; border-radius: 5px; padding: 10px; border: 2px solid #00e6e6; box-shadow: 0 0 15px #00e6e6;}
    .phase-inactive { background-color: transparent; color: #444; border-radius: 5px; padding: 10px; border: 2px solid #333;}

    /* Estil personalitzat per les pestanyes superiors */
    .stTabs [data-baseweb="tab-list"] { gap: 24px; }
    .stTabs [data-baseweb="tab"] { height: 50px; background-color: #111; border-radius: 5px 5px 0 0; padding: 10px; }
    .stTabs [aria-selected="true"] { background-color: #003333; border-bottom: 3px solid #00e6e6; color: #00e6e6 !important; }
    </style>
```

4. Generació de dades (telemetria simulada)

El sistema de moment no rep dades reals, sinó que les simula matemàticament amb la funció `generar_telemetria(t)`:

```python
# Funció matemàtica per simular la telemetria real del CanSat
def generar_telemetria(t):
    # Lògica d'ascens (fins al segon 100)
    if t < 100:
        alt = 450 + (t * 5.2) + np.random.normal(0, 0.5) # Puja a 5.2m/s amb una mica de soroll
        vel = 5.2
        pitch, roll = np.random.normal(0, 5), np.random.normal(0, 5) # Estable durant la pujada
    # Lògica de descens (després del segon 100)
    else:
        alt = max(450, 970 - ((t-100) * 1.5) + np.random.normal(0, 0.2)) # Baixa a 1.5m/s
        vel = -1.5 if alt > 450 else 0
        pitch, roll = np.random.normal(0, 15), np.random.normal(0, 15) # Més inestable al baixar

    temp = 22 - (alt * 0.006)             # La temperatura baixa amb l'alçada (gradient tèrmic)
    press = 1013 - (alt * 0.11)            # La pressió baixa mentre pugem
    co2 = 410 + np.random.randint(-5, 15)  # Nivells de CO2 simulats en ppm
    bat = max(0.0, 100 - (t * 0.05))       # Consum progressiu de la bateria
    rssi = -40 - (alt * 0.02) + np.random.normal(0, 2) # Pèrdua de senyal ràdio per distància
    yaw = (t * 10) % 360                   # Simulació de gir sobre l'eix vertical

    # Retorna un diccionari amb tots els valors calculats
    return {
        's': t, 'alt': round(alt, 1), 'temp': round(temp, 1), 'press': round(press, 1),
        'co2': co2, 'lat': 41.5644 + (t/100000), 'lon': 2.0006 + (t/100000),
        'vel': round(vel, 1), 'bat': round(bat, 1), 'rssi': round(rssi, 1),
        'pitch': round(pitch, 1), 'roll': round(roll, 1), 'yaw': round(yaw, 1)
    }
```

- **Altitud:** puja fins a l'apogeu i després baixa
- **Velocitat:** positiva en ascens, negativa en descens
- **Temperatura i pressió:** disminueixen amb l'altura
- **CO_2, bateria i senyal:** valors simulats amb variació aleatòria
- **Orientació (pitch, roll, yaw):** simulació de moviments

També es guarda un **historial de dades** amb Pandas.

5. Barra lateral i sistema d'alertes

A la barra lateral es mostren:

```python
# --- 4. BARRA LATERAL (SIDEBAR) I SISTEMA D'ALARMES ---
with st.sidebar:
    # Mostra el logo de l'escola o un emoji si no troba el fitxer
    st.image("Logo.png", width=150) if os.path.exists("Logo.png") else st.write("🛰️")
    st.markdown("## 🚨 ESTAT DEL SISTEMA")

    # Crea un espai buit que s'omplirà dinàmicament amb les alertes
    alerta_placeholder = st.empty()

    st.divider()
    st.markdown("### 🛠️ CONFIGURACIÓ")
    st.checkbox("📡 Ràdio Telemetria", value=True) # Simulació d'activació de ràdio
    st.checkbox("💾 Gravar a la SD", value=True)    # Simulació de gravació local
    if st.button("♻️ Netejar Dades"):               # Botó per reiniciar l'històric
        st.session_state.history = st.session_state.history.iloc[0:0] # Esborra el DataFrame
        st.rerun()                                 # Torna a carregar l'aplicació
```

- Estat del sistema (bateria i ràdio)
- Alertes visuals:
    - bateria crítica
    - senyal dèbil
    - paracaigudes desplegat
- Opcions de configuració (checkbox)
- Botó per reiniciar dades

6. Capçalera del dashboard

La capçalera inclou:

```python
# --- 5. CAPÇALERA PRINCIPAL ---
cl, cc, cr = st.columns([1, 4, 1]) # Divideix la part superior en 3 columnes
with cl:
    if os.path.exists("Logo Can.png"): st.image("Logo Can.png", width=90) # Logo CanSat
with cc:
    # Títol principal amb estil HTML
    st.markdown("<h1 style='text-align: center; margin-bottom:0;'>SATPI-2026 | MISSION CONTROL</h1>", unsafe_allo
    st.markdown("<p style='text-align: center; color: #a0a0a0;'>🚀 CENTRE DE DADES AVANÇAT - IES BERNAT EL FERRER
with cr:
    st.write("") # Columna buida per mantenir el títol centrat
```

- Logo del projecte
- Títol principal (Mission Control)
- Subtítol descriptiu

S'organitza amb columnes per mantenir el disseny centrat i professional.

7. Bucle principal (core del sistema)

El programa funciona amb un **bucle que s'executa contínuament**:

- Genera noves dades cada iteració
- Actualitza l'historial
- Refresca els gràfics
- Simula el temps real

```
# --- 6. BUCLE PRINCIPAL (EL COR DEL DASHBOARD) ---
# Contenidor principal que s'actualitza a cada segon
main_placeholder = st.empty()

# Executa el bucle des d'on s'havia quedat fins al segon 800
for i in range(len(st.session_state.history), 800):
    data = generar_telemetria(i) # Genera dades del segon actual
    # Afegeix les noves dades al registre històric
    st.session_state.history = pd.concat([st.session_state.history, pd.DataFrame([data])], ignore_index=True)
    df = st.session_state.history.tail(60) # Selecciona només els últims 60 punts pels gràfics
    df_all = st.session_state.history      # Selecciona totes les dades per la ruta 3D

    # Lògica per decidir quina fase de la missió estem vivint
    if i < 5: fase_act = 0                    # Primers segons: Pre-vol
    elif i < 98: fase_act = 1                 # Mentre pugem: Ascens
    elif 98 <= i < 105: fase_act = 2          # Al punt més alt: Apogeu
    elif data['alt'] > 450.5: fase_act = 3    # Mentre baixem: Descens
    else: fase_act = 4                        # Si estem a terra: Aterratge
```

També determina la fase del vol:

1. Pre-vol
2. Ascens
3. Apogeu
4. Descens
5. Aterratge

8. Visualització de dades (tabs)

El dashboard es divideix en tres pestanyes:

Visió general

- Temps, altitud, velocitat
- Gràfic d'elevació
- Mapa GPS

línia 185 a la 206

```
# --- FILA 2: PESTANYES INTERACTIVES (Tabs) ---
tab1, tab2, tab3 = st.tabs(["🛰️ VISIÓ GENERAL", "🌦️ ANÀLISI ATMOSFÈRIC", "🛞 NAVEGACIÓ I SISTEMES"])

# PESTANYA 1: Dades bàsiques i telemetria
with tab1:
    m1, m2, m3, m4 = st.columns(4) # Quatre columnes per mètriques clau
    m1.markdown(f"<div class='metric-card'><div class='metric-title'>TEMPS (T+)</div><div class='metric-value'>{data['s']}s</div></div>", unsafe_allow_html=True)
    m2.markdown(f"<div class='metric-card'><div class='metric-title'>ALTITUD</div><div class='metric-value'>{data['alt']}m</div></div>", unsafe_allow_html=True)
    m3.markdown(f"<div class='metric-card'><div class='metric-title'>VELOCITAT VERT.</div><div class='metric-value'>{data['vel']}m/s</div></div>", unsafe_allow_html=True)
    m4.markdown(f"<div class='metric-card'><div class='metric-title'>DISTÀNCIA</div><div class='metric-value'>{(data['s']*2.1):.1f}m</div></div>", unsafe_allow_html=True)

    g1, g2 = st.columns([2, 1]) # Gràfics de visió general
    with g1:
        # Gràfic de línia amb àrea plena per l'altitud
        fig_alt = go.Figure(go.Scatter(x=df['s'], y=df['alt'], fill='tozeroy', line=dict(color='#00e6e6', width=3)))
        fig_alt.update_layout(title="Perfil d'Elevació", template="plotly_dark", paper_bgcolor='rgba(0,0,0,0)', height=350)
        st.plotly_chart(fig_alt, use_container_width=True)
    with g2:
        # Mapa 2D de la posició GPS
        fig_map = px.scatter_mapbox(df, lat="lat", lon="lon", zoom=15, mapbox_style="carto-darkmatter")
        fig_map.update_traces(marker=dict(size=12, color="#00e6e6"))
        fig_map.update_layout(title="Posició GPS", margin={"r":0,"t":40,"l":0,"b":0}, height=350, paper_bgcolor='rgba(0,0,0,0)')
        st.plotly_chart(fig_map, use_container_width=True)
```

9. Actualització en temps real

El sistema es refresca cada **0.5 segons**, creant una simulació fluida i contínua del vol.

```
                    fig_3d.update_layout(scene=dict(bgcolor="rgba(0,0,0,0)"), template="plotly_dark", height=350, margin=dict(l=0,r=0,
                    st.plotly_chart(fig_3d, use_container_width=True)
                else:
                    st.info("Esperant dades per generar mapa 3D...") # Missatge si encara no hi ha prou punts

    time.sleep(0.5) # Espera mig segon abans de la següent actualització
```