```
import streamlit as st
import json
import os
import time

# ============================================================
# PATHS
# ============================================================

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
LOG_DIR = os.path.join(BASE_DIR, "logs")

LOG_FILE = os.path.join(LOG_DIR, "events.jsonl")
APPROVED_RULES = os.path.join(LOG_DIR, "approved_rules.conf")
REJECTED_FILE = os.path.join(LOG_DIR, "rejected_events.json")

os.makedirs(LOG_DIR, exist_ok=True)

if not os.path.exists(REJECTED_FILE):
    with open(REJECTED_FILE, "w") as f:
        json.dump([], f)

open(APPROVED_RULES, "a").close()
open(LOG_FILE, "a").close()

# ============================================================
# STREAMLIT CONFIG
# ============================================================

st.set_page_config(page_title="ML-Enabled WAF Dashboard", layout="wide")

# ============================================================
# CSS — REAL CARD (IMPORTANT)
# ============================================================

st.markdown("""
<style>
body {
    background-color: #0e1117;
}

/* Center content */
.main > div {
    max-width: 980px;
    margin: auto;
}

/* 🔥 REAL CARD — ADJACENT SIBLING FIX */
.rule-card-anchor + div[data-testid="stVerticalBlock"] {
    background: #e6f4ea !important;
    border: 2px solid #34a853 !important;
    border-radius: 18px;
    padding: 1.6rem;
    margin-bottom: 1.6rem;
    box-shadow: 0 10px 24px rgba(0,0,0,0.25);
}

/* Badge */
.rule-badge {
    display: inline-block;
    padding: 4px 12px;
    border-radius: 999px;
    background: #7c2d12;
    color: white;
    font-size: 12px;
    margin-bottom: 10px;
}

/* Metrics */
.metric {
    background: #020617;
    border: 1px solid #1f2937;
    border-radius: 14px;
    padding: 1.2rem;
    text-align: center;
}
</style>
""", unsafe_allow_html=True)

# ==========================================================
# HEADER
# ==========================================================

st.markdown("""
# 🛡 ML-Enabled Web Application Firewall
### Actionable Rule Recommendation Queue
""")

st.divider()

# ==========================================================
# UTILITIES
# ==========================================================

def read_jsonl(path):
    out = []
    with open(path, "r", encoding="utf-8") as f:
        for line in f:
            if line.strip():
                try:
                    obj = json.loads(line)
                    if isinstance(obj, dict):
                        out.append(obj)
                except json.JSONDecodeError:
                    pass
    return out

def load_rejected():
    with open(REJECTED_FILE) as f:
        return set(json.load(f))

def save_rejected(ids):
    with open(REJECTED_FILE, "w") as f:
        json.dump(list(ids), f, indent=2)

def approve_rule(rule):
    with open(APPROVED_RULES, "a") as f:
        f.write(rule + "\n")

def rule_id(e, r):
    return f"{e['timestamp']}_{e['uri']}_{hash(r)}"

# ==========================================================
# LOAD DATA
# ==========================================================

events = read_jsonl(LOG_FILE)
rejected_ids = load_rejected()

cards = []
for e in events:
    for r in e.get("recommended_rules", []):
        rid = rule_id(e, r)
        if rid not in rejected_ids:
            cards.append({
                "rid": rid,
                "uri": e["uri"],
                "timestamp": e["timestamp"],
                "reasons": e.get("reasons", []),
                "rule": r
            })

approved_count = sum(1 for _ in open(APPROVED_RULES) if _.strip())
rejected_count = len(rejected_ids)

# ==========================================================
# METRICS
# ==========================================================

c1, c2, c3 = st.columns(3)
```

```
with c1:
    st.markdown(f"<div class='metric'><h2>{len(cards)}</h2>Pending</div>",
unsafe_allow_html=True)
with c2:
    st.markdown(f"<div class='metric'><h2>{approved_count}</h2>Approved</div>",
unsafe_allow_html=True)
with c3:
    st.markdown(f"<div class='metric'><h2>{rejected_count}</h2>Rejected</div>",
unsafe_allow_html=True)

st.divider()

# =========================================================
# REAL RECTANGULAR RULE CARDS
# =========================================================

if not cards:
    st.success("✅ No actionable ML rule recommendations")
else:
    for c in reversed(cards[-20:]):

        with st.container():
            # invisible anchor for CSS selector
            st.markdown("<div class='rule-card-anchor'></div>",
unsafe_allow_html=True)

            st.markdown("<span class='rule-badge'>ML RECOMMENDATION</span>",
unsafe_allow_html=True)
            st.markdown(f"### 🔗 `{c['uri']}`")
            st.caption(f"🕒 {c['timestamp']}")

            st.markdown("#### 🧠 Explainability")
            for r in c["reasons"]:
                st.markdown(f"- {r}")

            st.markdown("#### 📜 Recommended ModSecurity Rule")
            st.code(c["rule"], language="apache")

            col1, col2 = st.columns(2)
            with col1:
                if st.button("✅ Approve Rule", key=f"a_{c['rid']}"):
                    approve_rule(c["rule"])
                    rejected_ids.add(c["rid"])
                    save_rejected(rejected_ids)
                    st.rerun()
            with col2:
                if st.button("❌ Reject", key=f"r_{c['rid']}"):
                    rejected_ids.add(c["rid"])
                    save_rejected(rejected_ids)
                    st.rerun()
```

```
# =========================================================
# AUTO REFRESH
# =========================================================

time.sleep(5)
st.rerun()
```