QAI Modular Socio-Economic Governance System with Middleware Fabric - Python Simulation

This document contains a Python-based demo simulation representing a modular QAI system with a middleware fabric. It demonstrates interactions between departments (Employment, Treasury, Licensing, HR), metadata roll-up, anomaly detection, alert propagation, and QAI-based conscious ethical override.

# 🔧 Python Simulation Code

import uuid  
from datetime import datetime  
  
# Define base components and logging  
audit\_log = []  
  
def log\_event(event):  
 timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  
 audit\_log.append(f"[{timestamp}] {event}")  
 print(f"[{timestamp}] {event}")  
  
# Department Modules  
class Candidate:  
 def \_\_init\_\_(self, name, license\_verified, skills\_score, salary\_expected, favored=False):  
 self.name = name  
 self.license\_verified = license\_verified  
 self.skills\_score = skills\_score  
 self.salary\_expected = salary\_expected  
 self.favored = favored  
  
class JobRole:  
 def \_\_init\_\_(self, title, min\_skills\_score, license\_required, salary\_band):  
 self.title = title  
 self.min\_skills\_score = min\_skills\_score  
 self.license\_required = license\_required  
 self.salary\_band = salary\_band  
  
# Middleware Module  
class MiddlewareFabric:  
 def \_\_init\_\_(self):  
 self.alerts = []  
  
 def verify\_license(self, candidate):  
 return candidate.license\_verified  
  
 def check\_skills(self, candidate, job):  
 return candidate.skills\_score >= job.min\_skills\_score  
  
 def check\_salary(self, candidate, job):  
 return job.salary\_band[0] <= candidate.salary\_expected <= job.salary\_band[1]  
  
 def generate\_alert(self, candidate, reason):  
 ticket\_id = f"TICKET-{uuid.uuid4().hex[:8]}"  
 alert = f"🚨 Alert: Candidate '{candidate.name}' flagged for: {reason}. Ticket ID: {ticket\_id}"  
 self.alerts.append(alert)  
 log\_event(alert)  
 return ticket\_id  
  
 def route\_to\_conscious\_agent(self, candidates, job, human\_choice):  
 qualified = [c for c in candidates if self.verify\_license(c) and  
 self.check\_skills(c, job) and self.check\_salary(c, job)]  
 if not qualified:  
 return None, "No qualified candidate"  
 best = max(qualified, key=lambda c: c.skills\_score)  
 if best.name != human\_choice.name:  
 log\_event(f"⚖️ QAI Override: Human selected '{human\_choice.name}' but best candidate is '{best.name}'")  
 return best, "QAI Overrode"  
 log\_event(f"✅ QAI Audit: Human choice '{human\_choice.name}' is correct")  
 return best, "QAI Approved"  
  
# Department Decision Logic  
def simulate\_department\_decisions():  
 job = JobRole("AI Ethics Director", 90, True, (150000, 200000))  
 candidates = [  
 Candidate("Dr. Neha Roy", True, 95, 180000),  
 Candidate("Mr. Arjun Mehta", False, 70, 190000, favored=True)  
 ]  
  
 # Step 1: Human HR selects a favored candidate  
 human\_choice = [c for c in candidates if c.favored][0]  
 log\_event(f"👤 Human HR Decision: Selected '{human\_choice.name}' for {job.title}")  
  
 # Step 2: Middleware checks  
 middleware = MiddlewareFabric()  
 if not middleware.verify\_license(human\_choice):  
 middleware.generate\_alert(human\_choice, "Unverified license")  
 if not middleware.check\_skills(human\_choice, job):  
 middleware.generate\_alert(human\_choice, "Insufficient skills")  
 if not middleware.check\_salary(human\_choice, job):  
 middleware.generate\_alert(human\_choice, "Salary mismatch")  
  
 # Step 3: Conscious Agent makes final decision  
 final\_candidate, result = middleware.route\_to\_conscious\_agent(candidates, job, human\_choice)  
 if final\_candidate:  
 log\_event(f"🎯 Final QAI Decision: Appointed '{final\_candidate.name}' via {result}")  
 else:  
 log\_event(f"❌ No suitable candidate found for {job.title}")  
  
 return audit\_log  
  
# Run the simulation  
simulate\_department\_decisions()