# LangGraph Orchestrator with Full Eligibility and Decision Workflow

import asyncio  
import pandas as pd  
from langchain.chains import GraphChain  
from typing import Dict, Any  
  
  
# Helper function to read Excel data  
async def read\_excel(file\_path, sheet\_name):  
 """Read data from an Excel file asynchronously."""  
 return pd.read\_excel(file\_path, sheet\_name=sheet\_name)  
  
  
# Helper function to parse summarized text file  
def parse\_summary\_file(file\_path: str) -> Dict[str, Any]:  
 """Parse a manually summarized text file to extract key claim details."""  
 details = {}  
 with open(file\_path, "r") as file:  
 for line in file:  
 if ":" in line:  
 key, value = line.split(":", 1)  
 details[key.strip()] = value.strip()  
 return {  
 "claim\_id": details.get("Claim Number", "").replace("\*", "").strip(),  
 "service\_date\_from": details.get("Service date from", "").strip(),  
 "service\_date\_to": details.get("Service date to", "").strip(),  
 "total\_charged": float(details.get("Total Charged amount", 0)),  
 "allowed\_amount": float(details.get("Total allowed amount", 0)),  
 "patient\_responsibility": float(details.get("Patients Responsibility", 0)),  
 "elevance\_responsibility": float(details.get("Elevance Responsibility", 0)),  
 "elevance\_paid": float(details.get("Total Elevance paid", 0)),  
 "provider\_name": details.get("Provider Name", "").strip(),  
 "member\_name": details.get("Member Name", "").strip(),  
 }  
  
  
# Member Eligibility Agent  
class MemberEligibilityAgent(GraphChain):  
 async def run(self, inputs: Dict[str, Any]):  
 claims\_df = await read\_excel(claims\_file, "Claims")  
 match = claims\_df[  
 (claims\_df["patient\_name"] == inputs["member\_name"])  
 & (claims\_df["service\_date\_from"] == inputs["service\_date\_from"])  
 & (claims\_df["service\_date\_to"] == inputs["service\_date\_to"])  
 ]  
 return {"eligible": not match.empty, "reason": "Member eligibility failed" if match.empty else "Success"}  
  
  
# Provider Eligibility Agent  
class ProviderEligibilityAgent(GraphChain):  
 async def run(self, inputs: Dict[str, Any]):  
 provider\_df = await read\_excel(provider\_file, "Provider file")  
 match = provider\_df[  
 (provider\_df["provider\_name"] == inputs["provider\_name"])  
 & (provider\_df["Contract\_from"] <= inputs["service\_date\_from"])  
 & (provider\_df["Contract\_to"] >= inputs["service\_date\_to"])  
 ]  
 return {"eligible": not match.empty, "reason": "Provider eligibility failed" if match.empty else "Success"}  
  
  
# Provider Network Check Agent  
class ProviderNetworkCheckAgent(GraphChain):  
 async def run(self, inputs: Dict[str, Any]):  
 network\_df = await read\_excel(provider\_network\_file, "Provider network file")  
 match = network\_df[  
 (network\_df["provider\_name"] == inputs["provider\_name"])  
 & (network\_df["Provider network status"] == "Active")  
 ]  
 return {"eligible": not match.empty, "reason": "Provider network inactive" if match.empty else "Success"}  
  
  
# Decision Agent  
class DecisionAgent(GraphChain):  
 async def run(self, inputs: Dict[str, Any]):  
 claims\_df = await read\_excel(claims\_file, "Claims")  
 claim\_row = claims\_df[claims\_df["DCN"] == inputs["claim\_id"]]  
  
 if claim\_row.empty:  
 return {"decision": "Failed", "reason": "Claim not found"}  
   
 row = claim\_row.iloc[0]  
 checks\_passed = (  
 row["Total\_charged\_amt"] == inputs["total\_charged"]  
 and row["Allowed\_amount"] == inputs["allowed\_amount"]  
 and row["patient\_responsibility"] == inputs["patient\_responsibility"]  
 )  
 if checks\_passed and row["Elevance\_paid"] < inputs["elevance\_responsibility"]:  
 return {"decision": "Adjust", "reason": "Adjustment Required"}  
 return {"decision": "No Adjust", "reason": "No Adjustment Needed"}  
  
  
# Adjustment Initiation Agent  
class AdjustmentAgent(GraphChain):  
 async def run(self, inputs: Dict[str, Any]):  
 print("Initiating Adjustment...")  
 await asyncio.sleep(1) # Simulate delay  
 return {"status": "Adjustment Created", "claim\_id": inputs["claim\_id"]}  
  
  
# File Paths  
claims\_file = "claims\_file.xlsx"  
provider\_file = "provider\_file.xlsx"  
provider\_network\_file = "provider\_network.xlsx"  
summary\_file = "iqt\_summary.txt"  
  
  
# Main workflow function  
async def orchestrate\_workflow():  
 # Parse summary file  
 inputs = parse\_summary\_file(summary\_file)  
  
 # Define agents  
 member\_agent = MemberEligibilityAgent(name="Member Eligibility Check")  
 provider\_agent = ProviderEligibilityAgent(name="Provider Eligibility Check")  
 network\_agent = ProviderNetworkCheckAgent(name="Provider Network Check")  
 decision\_agent = DecisionAgent(name="Decision Check")  
 adjustment\_agent = AdjustmentAgent(name="Adjustment Initiation")  
  
 # Run eligibility checks concurrently  
 eligibility\_results = await asyncio.gather(  
 member\_agent.run(inputs), provider\_agent.run(inputs), network\_agent.run(inputs)  
 )  
 if any(not result["eligible"] for result in eligibility\_results):  
 print("Eligibility Check Failed:", eligibility\_results)  
 return  
  
 # Run decision agent  
 decision\_result = await decision\_agent.run(inputs)  
 print("Decision Result:", decision\_result)  
 if decision\_result["decision"] == "Adjust":  
 adjustment\_result = await adjustment\_agent.run(inputs)  
 print("Adjustment Result:", adjustment\_result)  
 else:  
 print("No Adjustment Required.")  
  
# Run the orchestrator  
asyncio.run(orchestrate\_workflow())