# LangGraph Orchestrator with Async Implementation (Final Version)

from langchain.chains import GraphChain  
from typing import Any, Dict  
import pandas as pd  
import asyncio  
  
# 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)  
  
  
# Eligibility Check Agent Class  
class EligibilityCheckAgent:  
 def \_\_init\_\_(self, name, sheet\_name, column\_name, check\_value):  
 self.name = name  
 self.sheet\_name = sheet\_name  
 self.column\_name = column\_name  
 self.check\_value = check\_value  
  
 async def execute(self, data):  
 """Perform the eligibility check asynchronously using Excel data."""  
 df = await read\_excel(excel\_file, sheet\_name=self.sheet\_name)  
 result = df[df[self.column\_name] == data[self.check\_value]]  
 if not result.empty:  
 return {"eligible": True, "details": result.to\_dict(orient="records")}  
 return {"eligible": False}  
  
  
# Decision Agent Class  
class DecisionAgent:  
 async def execute(self, data):  
 """Standalone pricing check asynchronously."""  
 df = await read\_excel(excel\_file, sheet\_name="Pricing")  
 result = df[df["ClaimID"] == data["claim\_id"]]  
 if not result.empty and result.iloc[0]["Price"] > 1000: # Assume $1000 threshold  
 return {"adjustment\_needed": True, "price": result.iloc[0]["Price"]}  
 return {"adjustment\_needed": False}  
  
  
# Adjustment Initiation Agent Class  
class AdjustmentInitiationAgent:  
 async def execute(self, data):  
 """Mock adjustment process asynchronously."""  
 print(f"Initiating adjustment for Claim ID: {data['claim\_id']}")  
 await asyncio.sleep(1) # Simulate delay  
 return {"status": "Adjustment Completed"}  
  
  
# Define the LangGraph Orchestrator Node  
class OrchestratorNode(GraphChain):  
 def \_\_init\_\_(self, eligibility\_agents, decision\_agent, adjustment\_agent):  
 self.eligibility\_agents = eligibility\_agents  
 self.decision\_agent = decision\_agent  
 self.adjustment\_agent = adjustment\_agent  
  
 async def execute(self, inquiry: Dict[str, Any]):  
 """Main execution logic for the Orchestrator."""  
 print("Processing IQT Document...")  
 summarized\_data = await self.process\_iqt(inquiry)  
 print(f"Summarized Data: {summarized\_data}")  
  
 print("Triggering Eligibility Check Agents in Parallel...")  
 eligibility\_results = await self.run\_eligibility\_checks(summarized\_data)  
  
 # Combine the results and determine overall eligibility  
 overall\_eligibility = all([result["eligible"] for result in eligibility\_results.values()])  
 print(f"Eligibility Results: {eligibility\_results}")  
   
 if not overall\_eligibility:  
 print("Eligibility failed. Closing IQT.")  
 return {"status": "IQT Closed", "reason": "Eligibility Failed"}  
  
 print("Triggering Decision Agent for Pricing Check...")  
 decision\_status = await self.decision\_agent.execute(summarized\_data)  
 print(f"Decision Status: {decision\_status}")  
  
 if decision\_status["adjustment\_needed"]:  
 print("Triggering Adjustment Initiation Agent...")  
 adjustment\_status = await self.adjustment\_agent.execute(summarized\_data)  
 print(f"Adjustment Status: {adjustment\_status}")  
 return adjustment\_status  
 else:  
 print("No adjustment required. Closing IQT.")  
 return {"status": "IQT Closed", "reason": "No Adjustment Needed"}  
  
 async def process\_iqt(self, inquiry):  
 """Mock function to process IQT documents and extract relevant details."""  
 return {"claim\_id": inquiry["claim\_id"], "provider\_id": "P123", "member\_id": "M456"}  
  
 async def run\_eligibility\_checks(self, data):  
 """Run eligibility checks in parallel using async."""  
 tasks = {agent.name: agent.execute(data) for agent in self.eligibility\_agents}  
 results = await asyncio.gather(\*tasks.values())  
 return dict(zip(tasks.keys(), results))  
  
  
# File path for the Excel file  
excel\_file = "claim\_data.xlsx" # Replace with your actual file path  
  
# Instantiate agents  
eligibility\_agents = [  
 EligibilityCheckAgent(  
 name="Provider Details Check", sheet\_name="Eligibility", column\_name="ProviderID", check\_value="provider\_id"  
 ),  
 EligibilityCheckAgent(  
 name="Provider Network Check", sheet\_name="Eligibility", column\_name="NetworkID", check\_value="provider\_id"  
 ),  
 EligibilityCheckAgent(  
 name="Member Eligibility Check", sheet\_name="Eligibility", column\_name="MemberID", check\_value="member\_id"  
 ),  
]  
  
decision\_agent = DecisionAgent()  
adjustment\_agent = AdjustmentInitiationAgent()  
  
# Define the workflow graph  
eligibility\_node = EligibilityCheckNode(name="Eligibility Check")  
pricing\_node = PricingCheckNode(name="Pricing Check")  
adjustment\_node = AdjustmentNode(name="Adjustment")  
  
# Connect nodes  
pricing\_node.add\_parent(eligibility\_node)  
adjustment\_node.add\_parent(pricing\_node)  
  
# Execute the workflow  
workflow = GraphChain(root=eligibility\_node)  
output = asyncio.run(workflow.run(inputs={"claim\_id": "C789"}))  
print(output)