# 🧠 Build Enterprise AI Agents with Google

**Collaborative Workshop - Orchestrators & Implementers Together**

### Event: The AI Collective North Dallas | 90 Minutes

**Everyone works together, everyone contributes!**

---

## 🎯 Workshop Philosophy

**Orchestrators** (non-coders) and **Implementers** (coders) work **side-by-side** throughout:

- Orchestrators design in AI Studio → Implementers code it immediately
- Implementers build features → Orchestrators test them live
- Continuous feedback loop → Better agent together!

**Goal**: Everyone leaves with a WORKING agent they built together.

---

## ⏰ 90-Minute Collaborative Timeline

| Time | Activity | Orchestrators Do | Implementers Do | Together |
|---|---|---|---|---|
| 0:00-0:15 | **Setup & Intro** | Get API key, open AI Studio | Clone repo, install deps | See final demo |
| 0:15-0:30 | **Build Agent Personality** | Design prompts in AI Studio | Code the system prompt | Test together |
| 0:30-0:50 | **Add Intelligence** | Create test cases | Build LangGraph workflow | Verify logic |
| 0:50-1:10 | **Add Knowledge** | Write FAQs | Implement RAG system | Test search |
| 1:10-1:25 | **Polish & Deploy** | Final testing | Deploy to Cloud Run | Celebrate! |
| 1:25-1:30 | **Wrap-up** | Share learnings | Share learnings | Q&A |

---

# Part 1: Setup & Introduction (15 min)

## Everyone: Get Your API Key (5 min)

**All participants**:

1. Go to: **https://aistudio.google.com/**

2. Click **"Get API Key"**
3. Create new key (FREE, no credit card!)
4. **Save it** - you'll need it in 2 minutes

---

## Implementers: Clone & Setup (5 min)

**While orchestrators explore AI Studio**, implementers set up:

```
# Clone the repository
git clone https://github.com/[YOUR-REPO]/insurance-agent-workshop
cd insurance-agent-workshop/backend

# Create virtual environment
python -m venv venv

# Activate it
# Windows:
venv\Scripts\activate
# Mac/Linux:
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Create .env file with your API key
echo "GEMINI_API_KEY=your_key_here" > .env
```

**Verify it works**:

```
python -c "import langchain; import langgraph; print('✅ Ready!')"
```

---

## Everyone: See the Final Product (5 min)

**Facilitator shows the complete working agent**:

- Natural conversation
- Remembers context
- Searches knowledge base
- Calculates quotes
- Explains reasoning

**This is what we're building together!**

---

# Part 2: Build Agent Personality (15 min)

## 🎨 Orchestrators: Design in AI Studio (10 min)

**Open AI Studio**: https://aistudio.google.com/

**Create a new "Freeform prompt"**:

```
You are "Alex", an expert insurance agent powered by AI.

Your personality:
- Friendly and professional
- Patient and helpful
- Explains complex terms simply
- Never pushy or salesy

Your role:
1. Help customers get accurate insurance quotes
2. Answer questions about coverage types
3. Gather required information conversationally

For AUTO insurance, you need:
- Customer's age
- Vehicle year, make, and model
- Years licensed
- Accident/violation history

Guidelines:
- Ask 1-2 questions at a time (don't overwhelm)
- When customers ask "what is X?", explain clearly
- When you have enough info, calculate the quote
- Always explain the breakdown

Start by greeting the customer warmly and asking what type of insurance they need.
```

**Test it in AI Studio**:

- Type: "Hi, I need insurance"
- See how Alex responds
- Iterate until you like the personality

**Share with implementers**: Read your prompt out loud!

---

## 💻 Implementers: Code the System Prompt (5 min)

**Create file**: `backend/system_prompt.py`

```
"""
System prompt designed by orchestrators in AI Studio
"""
```

```
INSURANCE_AGENT_PROMPT = """
You are "Alex", an expert insurance agent powered by AI.

Your personality:
- Friendly and professional
- Patient and helpful
- Explains complex terms simply
- Never pushy or salesy

Your role:
1. Help customers get accurate insurance quotes
2. Answer questions about coverage types
3. Gather required information conversationally

For AUTO insurance, you need:
- Customer's age
- Vehicle year, make, and model
- Years licensed
- Accident/violation history

For HOME insurance, you need:
- Year home was built
- Square footage
- Construction type
- Desired dwelling coverage

Guidelines:
- Ask 1-2 questions at a time (don't overwhelm)
- When customers ask "what is X?", explain clearly
- When you have enough info, calculate the quote
- Always explain the breakdown

Remember previous messages in the conversation. Never ask for information the user
already provided.
"""

def get_system_prompt():
    return INSURANCE_AGENT_PROMPT
```

**Save and commit**: `git add system_prompt.py && git commit -m "Add orchestrator-designed prompt"`

---

## ✅ Together: Test It! (5 min)

**Implementers**: Run a quick test:

```
# test_prompt.py
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.schema import SystemMessage, HumanMessage
from system_prompt import get_system_prompt
```

```
import os

llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",
    google_api_key=os.getenv("GEMINI_API_KEY")
)

messages = [
    SystemMessage(content=get_system_prompt()),
    HumanMessage(content="Hi, I need insurance")
]

response = llm.invoke(messages)
print(response.content)
```

**Run it**: `python test_prompt.py`

**Orchestrators**: Does the response match your design? If not, refine in AI Studio!

---

# Part 3: Add Intelligence (20 min)

## 🎨 Orchestrators: Create Test Cases (10 min)

**In AI Studio, design 5 test scenarios**:

**Test 1: Happy Path**

```
User says: "I'm 28, drive a 2020 Honda Civic, licensed 10 years, no accidents"
Expected: Agent should calculate quote immediately
Pass criteria: Returns monthly premium
```

**Test 2: Missing Information**

```
User says: "I need car insurance"
Expected: Agent asks for age, vehicle, and history
Pass criteria: Asks 1-2 questions, not all at once
```

**Test 3: Knowledge Question**

```
User says: "What's the difference between collision and comprehensive?"
Expected: Agent explains both coverage types
Pass criteria: Clear explanation of both
```

**Test 4: Partial Information**

```
User says: "I'm 28 and drive a Honda"
Expected: Agent asks for vehicle year and driving history
Pass criteria: Doesn't repeat asking for age
```

**Test 5: Multi-Turn Conversation**

```
Turn 1: "I need car insurance"
Turn 2: "I'm 28"
Turn 3: "I drive a 2020 Honda Civic"
Expected: Agent remembers age from Turn 2
Pass criteria: Doesn't ask for age again
```

**Share these with implementers** - they'll code the logic!

---

## 💻 Implementers: Build LangGraph Workflow (10 min)

**Create file**: `backend/langgraph_agent.py`

```python
"""
LangGraph agent workflow - implements orchestrator test cases
"""

from typing import TypedDict, Sequence, Annotated
from langgraph.graph import StateGraph, END
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.schema import HumanMessage, AIMessage, SystemMessage
from langchain.tools import tool
import operator
import os

# Agent State (Memory)
class AgentState(TypedDict):
    """State that flows through the graph"""
    messages: Annotated[Sequence[HumanMessage | AIMessage], operator.add]
    user_info: dict  # Extracted: age, vehicle, etc.
    insurance_type: str  # 'auto' or 'home'
    quote_result: dict
    knowledge_context: str
    next_action: str

# Initialize LLM
llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",
    google_api_key=os.getenv("GEMINI_API_KEY"),
    temperature=0.7
)
```

```
# Tool: Calculate Premium
@tool
def calculate_auto_premium(
    age: int,
    vehicle_year: int,
    years_licensed: int,
    accidents: int = 0,
    violations: int = 0
) -> dict:
    """Calculate auto insurance premium based on driver profile"""

    base_rate = 800

    # Age factor
    if age < 25:
        age_factor = 400
    elif age < 30:
        age_factor = 200
    else:
        age_factor = 0

    # Experience discount
    experience_factor = -100 if years_licensed > 10 else 0

    # Accident/violation surcharges
    accident_factor = accidents * 300
    violation_factor = violations * 200

    annual_premium = (base_rate + age_factor + experience_factor +
                     accident_factor + violation_factor)
    monthly_premium = round(annual_premium / 12, 2)

    return {
        "monthly_premium": monthly_premium,
        "annual_premium": annual_premium,
        "breakdown": {
            "base_rate": base_rate,
            "age_adjustment": age_factor,
            "experience_discount": experience_factor,
            "accident_surcharge": accident_factor,
            "violation_surcharge": violation_factor
        }
    }

# Node 1: Gather Information
def gather_info_node(state: AgentState) -> AgentState:
    """Gather information through conversation"""
    print("🟢 NODE: gather_info")

    from system_prompt import get_system_prompt

    # Build messages for LLM
    messages = [SystemMessage(content=get_system_prompt())]
    messages.extend(state["messages"])
    # Get LLM response
    response = llm.invoke(messages)

    # Add to conversation
    state["messages"].append(AIMessage(content=response.content))

    # Simple extraction (in production, use structured output)
    last_user_msg = state["messages"][-2].content if len(state["messages"]) > 1
else ""

    # Extract info from user message
    if "age" not in state["user_info"] and any(str(i) in last_user_msg for i in
range(18, 100)):
        # Extract age
        for i in range(18, 100):
            if str(i) in last_user_msg:
                state["user_info"]["age"] = i
                break

    # Extract vehicle year
    if "vehicle_year" not in state["user_info"]:
        for year in range(2000, 2026):
            if str(year) in last_user_msg:
                state["user_info"]["vehicle_year"] = year
                break

    # Extract years licensed
    if "years_licensed" not in state["user_info"]:
        if "10 years" in last_user_msg or "licensed for 10" in last_user_msg:
            state["user_info"]["years_licensed"] = 10

    # Extract accidents
    if "accidents" not in state["user_info"]:
        if "no accidents" in last_user_msg.lower():
            state["user_info"]["accidents"] = 0

    state["next_action"] = "check_if_ready"
    return state

# Node 2: Calculate Quote
def calculate_quote_node(state: AgentState) -> AgentState:
    """Calculate insurance quote using tools"""
    print("🟢 NODE: calculate_quote")

    user_info = state.get("user_info", {})
    result = calculate_auto_premium.invoke(user_info)

    state["quote_result"] = result
    state["next_action"] = "explain_quote"

    print(f"   💰 Premium: ${result['monthly_premium']}/month")

    # Add explanation to conversation
    explanation = f"""
Based on your information, here's your quote:

**Monthly Premium: ${result['monthly_premium']}**
**Annual Premium: ${result['annual_premium']}**

Breakdown:
- Base rate: ${result['breakdown']['base_rate']}
- Age adjustment: ${result['breakdown']['age_adjustment']}
- Experience discount: ${result['breakdown']['experience_discount']}
- Accident surcharge: ${result['breakdown']['accident_surcharge']}

This quote is valid for 30 days. Would you like to proceed or have any questions?
"""

    state["messages"].append(AIMessage(content=explanation))

    return state

# Conditional Edge: Decide Next Step
def should_calculate(state: AgentState) -> str:
    """Decide if we should calculate or gather more info"""

    user_info = state.get("user_info", {})

    # Check if we have minimum required info
    required_fields = ["age", "vehicle_year", "years_licensed"]

    if all(field in user_info for field in required_fields):
        return "calculate"
    else:
        return "gather_more"

# Build the Graph
workflow = StateGraph(AgentState)

# Add nodes
workflow.add_node("gather_info", gather_info_node)
workflow.add_node("calculate_quote", calculate_quote_node)

# Set entry point
workflow.set_entry_point("gather_info")

# Add conditional edges
workflow.add_conditional_edges(
    "gather_info",
    should_calculate,
    {
        "calculate": "calculate_quote",
        "gather_more": "gather_info"  # Loop back
    }
)

# Add edge to end
```

```
workflow.add_edge("calculate_quote", END)

# Compile
agent_graph = workflow.compile()

# Test function
if __name__ == "__main__":
    # Test with orchestrator's Test Case 1
    result = agent_graph.invoke({
        "messages": [
            HumanMessage(content="I'm 28, drive a 2020 Honda Civic, licensed 10
years, no accidents")
        ],
        "user_info": {},
        "insurance_type": "auto"
    })

    print("\n" + "="*50)
    print("FINAL CONVERSATION:")
    print("="*50)
    for msg in result["messages"]:
        role = "User" if isinstance(msg, HumanMessage) else "Agent"
        print(f"\n{role}: {msg.content}")
```

**Run it**: `python langgraph_agent.py`

---

## ☑ Together: Verify Test Cases (5 min)

**Orchestrators**: Run your test cases!

**Test 1**:

```
python langgraph_agent.py
```

Expected: Should calculate quote immediately ☑

**Implementers**: Create a test script for the other cases:

```
# test_cases.py
from langgraph_agent import agent_graph
from langchain.schema import HumanMessage

# Test 2: Missing information
print("TEST 2: Missing information")
result = agent_graph.invoke({
    "messages": [HumanMessage(content="I need car insurance")],
    "user_info": {},
    "insurance_type": "auto"
})
print(result["messages"][-1].content)

# Test 4: Partial information
print("\nTEST 4: Partial information")
result = agent_graph.invoke({
    "messages": [HumanMessage(content="I'm 28 and drive a Honda")],
    "user_info": {},
    "insurance_type": "auto"
})
print(result["messages"][-1].content)
```

**Orchestrators**: Check if responses match your expectations!

---

# Part 4: Add Knowledge (20 min)

## 🎨 Orchestrators: Write Insurance FAQs (10 min)

**In AI Studio, create 10 FAQs**:

```
Category: Auto Coverage

Q: What is collision coverage?
A: Collision coverage pays for damage to YOUR vehicle when you hit another vehicle
or object, regardless of who's at fault.

Q: What is comprehensive coverage?
A: Comprehensive coverage pays for damage to your vehicle from non-collision
events like theft, vandalism, weather, fire, or animal strikes.

Q: What is liability coverage?
A: Liability coverage pays for damage YOU cause to others, including bodily injury
and property damage. It's required by law in most states.

Category: Discounts

Q: How can I lower my premium?
A: Common ways include: bundling policies (10-25% off), maintaining a clean
driving record (15-30% off), installing safety features, and increasing your
deductible.

Q: Do I get a discount for being a good driver?
A: Yes! No accidents or violations in 3+ years can save you 15-30% on your
premium.

Category: General

Q: What is a deductible?
A: A deductible is the amount you pay out-of-pocket before your insurance coverage
```

```
kicks in. Higher deductibles = lower premiums.

Q: Why is insurance more expensive for young drivers?
A: Statistically, drivers under 25 have more accidents, so insurers charge higher
premiums to offset the risk.

Q: Can I cancel my policy anytime?
A: Yes, but you may face a cancellation fee. It's best to wait until your policy
renewal date.

Q: What's the difference between full coverage and liability only?
A: Full coverage includes liability + collision + comprehensive. Liability only
covers damage you cause to others, not your own vehicle.

Q: Do I need rental car coverage?
A: It depends. If you have another vehicle or can easily get a ride, you may not
need it. Otherwise, it's worth the $5-10/month.
```

**Share with implementers**: Read these out loud or paste in chat!

---

## 💻 Implementers: Build RAG System (10 min)

**Create file**: `backend/rag_system.py`

```python
"""
RAG (Retrieval Augmented Generation) system
Uses orchestrator-created FAQs as knowledge base
"""

from langchain_chroma import Chroma
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain.schema import Document
import os

# Insurance knowledge base (from orchestrators!)
INSURANCE_KNOWLEDGE = {
    "auto_coverage": [
        "Collision coverage pays for damage to YOUR vehicle when you hit another
vehicle or object, regardless of who's at fault.",
        "Comprehensive coverage pays for damage to your vehicle from non-collision
events like theft, vandalism, weather, fire, or animal strikes.",
        "Liability coverage pays for damage YOU cause to others, including bodily
injury and property damage. It's required by law in most states."
    ],
    "discounts": [
        "Common ways to lower your premium include: bundling policies (10-25%
off), maintaining a clean driving record (15-30% off), installing safety features,
and increasing your deductible.",
        "No accidents or violations in 3+ years can save you 15-30% on your
premium."
```

```
    ],
    "general": [
        "A deductible is the amount you pay out-of-pocket before your insurance coverage kicks in. Higher deductibles = lower premiums.",
        "Drivers under 25 have statistically more accidents, so insurers charge higher premiums to offset the risk.",
        "You can cancel your policy anytime, but you may face a cancellation fee. It's best to wait until your policy renewal date.",
        "Full coverage includes liability + collision + comprehensive. Liability only covers damage you cause to others, not your own vehicle.",
        "Rental car coverage costs $5-10/month. It's worth it if you don't have another vehicle or can't easily get a ride."
    ]
}

# Initialize embeddings
embeddings = GoogleGenerativeAIEmbeddings(
    model="models/embedding-001",
    google_api_key=os.getenv("GEMINI_API_KEY")
)

def initialize_knowledge_base():
    """Create vector store from orchestrator FAQs"""

    documents = []
    for category, items in INSURANCE_KNOWLEDGE.items():
        for item in items:
            documents.append(Document(
                page_content=item,
                metadata={"category": category}
            ))

    vectorstore = Chroma.from_documents(
        documents,
        embeddings,
        persist_directory="./insurance_knowledge_db"
    )

    print(f"✅ Knowledge base initialized with {len(documents)} documents")
    return vectorstore

def search_knowledge(query: str, k: int = 2):
    """Search knowledge base for relevant information"""

    try:
        vectorstore = Chroma(
            persist_directory="./insurance_knowledge_db",
            embedding_function=embeddings
        )
    except:
        # First time - create it
        vectorstore = initialize_knowledge_base()

    results = vectorstore.similarity_search(query, k=k)
    return results

def get_relevant_context(query: str, insurance_type: str = None) -> str:
    """Get relevant context as a string"""

    results = search_knowledge(query, k=2)
    context = "\n\n".join([doc.page_content for doc in results])
    return context

# Initialize on import
if __name__ == "__main__":
    vectorstore = initialize_knowledge_base()

    # Test with orchestrator's FAQ
    print("\nTesting search:")
    results = search_knowledge("What is collision coverage?")
    for doc in results:
        print(f"- {doc.page_content}")
```

**Run it**: `python rag_system.py`

---

### ✅ Together: Test Knowledge Search (5 min)

**Orchestrators**: Ask your FAQ questions!

```
# test_rag.py
from rag_system import search_knowledge

# Test orchestrator's questions
questions = [
    "What is collision coverage?",
    "How can I lower my premium?",
    "What is a deductible?"
]

for q in questions:
    print(f"\nQ: {q}")
    results = search_knowledge(q, k=1)
    print(f"A: {results[0].page_content}")
```

**Run it**: `python test_rag.py`

**Orchestrators**: Are the answers correct? If not, refine the FAQs!

---

## Part 5: Polish & Deploy (15 min)

### 💻 Implementers: Create FastAPI Backend (10 min)

**Create file**: `backend/main.py`

```
"""
FastAPI backend - brings everything together
"""

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
import uuid
from langgraph_agent import agent_graph
from langchain.schema import HumanMessage
import os

app = FastAPI(
    title="Agentic Insurance Agent",
    description="Built by orchestrators & implementers together!"
)

# CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Session storage
sessions = {}

class ChatRequest(BaseModel):
    message: str
    session_id: str = None

class ChatResponse(BaseModel):
    response: str
    session_id: str

@app.post("/api/chat", response_model=ChatResponse)
async def chat(request: ChatRequest):
    """Chat with the agent"""

    session_id = request.session_id or str(uuid.uuid4())

    # Initialize session
    if session_id not in sessions:
        sessions[session_id] = {
            "messages": [],
            "user_info": {},
            "insurance_type": None
        }
    session = sessions[session_id]

    # Add user message
    session["messages"].append(HumanMessage(content=request.message))

    # Run through agent graph
    result = agent_graph.invoke(session)

    # Update session
    sessions[session_id] = result

    # Get agent response
    agent_response = result["messages"][-1].content

    return ChatResponse(
        response=agent_response,
        session_id=session_id
    )

@app.get("/health")
async def health():
    return {"status": "healthy", "message": "Agent is ready!"}

if __name__ == "__main__":
    import uvicorn
    print("\n" + "="*50)
    print("🚀 Starting Agentic Insurance Agent")
    print("="*50)
    print("\n📊 Built by:")
    print("  - Orchestrators: Prompts, test cases, FAQs")
    print("  - Implementers: LangGraph, RAG, FastAPI")
    print("\n🌐 API running at: http://localhost:8000")
    print("📚 Docs at: http://localhost:8000/docs")
    print("\n" + "="*50 + "\n")

    uvicorn.run(app, host="0.0.0.0", port=8000)
```

**Run it**: `python main.py`

---

## 🎨 Orchestrators: Final Testing (5 min)

**Test the complete agent**:

```
# Test 1: Happy path
curl -X POST http://localhost:8000/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message": "I'\''m 28, drive a 2020 Honda Civic, licensed 10 years, no accidents"}'

# Test 2: Knowledge question
```

```
curl -X POST http://localhost:8000/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message": "What is collision coverage?"}'

# Test 3: Multi-turn conversation
curl -X POST http://localhost:8000/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message": "I need car insurance", "session_id": "test123"}'

curl -X POST http://localhost:8000/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message": "I'\''m 28", "session_id": "test123"}'
```

**Check**: Do all your test cases pass?

---

## 💻 Implementers: Deploy to Cloud Run (5 min)

**Create** `Dockerfile`:

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

**Deploy**:

```
gcloud run deploy insurance-agent \
  --source ./backend \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars GEMINI_API_KEY=$GEMINI_API_KEY
```

**Get URL**: https://insurance-agent-xxx.run.app

**Everyone**: Test the live agent!

---

# Part 6: Wrap-Up (5 min)

## 🎉 What We Built Together

**Orchestrators contributed**:

- ✅ Agent personality (system prompt)
- ✅ 5 test cases
- ✅ 10 insurance FAQs
- ✅ Quality assurance

**Implementers built**:

- ✅ LangGraph workflow
- ✅ RAG system with vector search
- ✅ FastAPI backend
- ✅ Cloud deployment

**Together we created**:

- ✅ A production-ready Agentic AI agent
- ✅ 100% free to run
- ✅ Demonstrates all 6 principles
- ✅ Scalable to enterprise

---

## 🎯 Key Learnings

1. **Collaboration works**: Non-coders and coders complement each other
2. **AI Studio bridges the gap**: Orchestrators prototype, implementers code
3. **Agentic AI = 6 principles**: Autonomy, Reasoning, Tools, Memory, Planning, Learning
4. **Google stack is powerful**: Free, fast, and production-ready
5. **Everyone can contribute**: You don't need to code to build AI!

---

## 📚 Resources

**Code Repository**: [Share your GitHub repo]

**Documentation**:

- Google AI Studio: https://aistudio.google.com/
- LangGraph: https://langchain-ai.github.io/langgraph/
- Cloud Run: https://cloud.google.com/run

**Community**:

- The AI Collective: [Your community link]
- Next workshop: [Date]

---

## 🚀 Next Steps

**For Orchestrators**:

- Refine prompts for different industries

- Create more comprehensive test suites
- Design knowledge bases for other domains

**For Implementers**:

- Add more insurance types (life, health)
- Implement document upload (Gemini Vision)
- Add monitoring and analytics
- Scale to production

**For Everyone**:

- Share your agent on social media
- Build agents for your own use cases
- Join the community
- Attend future workshops!

---

**Thank you for building together!** 🎉

**Questions?**

---

## Appendix: Complete File Structure

```
insurance-agent-workshop/
├── backend/
│   ├── .env                    # Your API key
│   ├── system_prompt.py        # ✅ Orchestrator-designed
│   ├── langgraph_agent.py      # ✅ Implementer-built
│   ├── rag_system.py           # ✅ Uses orchestrator FAQs
│   ├── main.py                 # ✅ FastAPI integration
│   ├── test_prompt.py          # Test system prompt
│   ├── test_cases.py           # Test orchestrator cases
│   ├── test_rag.py             # Test knowledge search
│   ├── requirements.txt        # Dependencies
│   └── insurance_knowledge_db/ # Vector store
│
├── Dockerfile                  # For deployment
└── README.md                   # Quick start guide
```

---

## Appendix: requirements.txt

```
langchain==1.1.0
langgraph==1.0.4
langchain-google-genai==3.2.0
langchain-chroma==0.1.0
chromadb==0.5.5
```

```
fastapi==0.123.0
uvicorn==0.38.0
python-dotenv==1.2.1
pydantic==2.12.5
```

---

**END OF COLLABORATIVE WORKSHOP GUIDE**