90-Minute Workshop – Insurance Agent Demo

Overview

This guide walks an audience through **downloading**, **setting up**, **running**, and **interacting** with the Insurance Agent application. It explains what happens under the hood, shows the LangGraph workflow, and describes how to verify backend state and deploy the app.

---

1. Prerequisites

Requirement	Why	Install Command
Python 3.12+	Backend runtime	`py -3.12 -m pip install --upgrade pip`
Node.js 20+	Front-end dev server	Download from https://nodejs.org
Git	Clone repo	`git clone <repo-url>`
Google Gemini API key	LLM access	Create a key in Google AI Studio and place it in a `.env` file (see step 2)

---

2. Clone the repository & set up environment

```
# Clone
git clone https://github.com/yourorg/insurance_agent.git
cd insurance_agent

# Backend – create virtual env (optional but recommended)
python -m venv venv
source venv/bin/activate   # on Windows: venv\Scripts\activate

# Install Python deps
pip install -r backend/requirements.txt

# Frontend – install npm packages
cd frontend
npm install
cd ..
```

Create a `.env` file in `backend/`:

```
GEMINI_API_KEY=YOUR_GEMINI_API_KEY_HERE
```

*(Do **not** commit this file to Git.)*

3. Run the services (manual, not via Antigravity)

3.1 Start the FastAPI backend

```
cd backend
uvicorn main:app --host 127.0.0.1 --port 8001
```

- The API is now reachable at `http://127.0.0.1:8001`.
- Health endpoint: `GET /health` → JSON with status, session count, etc.

3.2 Start the React/Vite frontend

```
cd ../frontend
npm run dev
```

- The UI will be served at `http://localhost:5173` (Vite default).
- The UI is already configured to call the backend on **port 8001** (see `ChatInterface.jsx`).

---

4. Interacting with the Chatbot

1. Open a browser and navigate to `http://localhost:5173`.
2. Type a message, e.g. **"I need auto insurance"**.
3. The UI sends a POST to `/api/chat`.
4. **What happens under the hood**:
   - `main.py` receives the request, stores the user message in an in-memory `sessions` dict.
   - **(Current state)** The request is processed by a **single Gemini LLM call** (the LangGraph brain is *not* invoked yet).
   - If the message contains a knowledge-question keyword (`what is`, `explain`, ...) the RAG system (`rag_system.search_knowledge`) is called **before** the LLM to inject relevant context.
   - The LLM generates the response, which is appended to the session and returned to the UI.
5. The UI displays the response. You can continue the conversation; the session state persists until you call **Reset** (`POST /api/reset`).

---

5. Verifying Backend Data

- Open a terminal and run:

```
curl http://127.0.0.1:8001/health
```

→ Shows `sessions` count.

- To inspect a specific session, add a temporary endpoint (or use the Python REPL):

```
import json, requests
s = requests.get('http://127.0.0.1:8001/health').json()
print(s)
```

- The `sessions` dict contains:
  - `messages` (list of `HumanMessage` / `AIMessage` objects)
  - `user_info` (collected details for premium calculation)
  - `insurance_type`, `quote_result`, `knowledge_context`, `next_action`.

---

6. LangGraph Integration

- The **LangGraph workflow** lives in `backend/langgraph_agent.py` and is visualised in `agent_architecture.md` (Mermaid diagram).
- **Current status**: The workflow is **separate** from the `/api/chat` endpoint. It can be exercised directly via the test script `test_langgraph_brain.py`.
- **Planned integration** (see `implementation_plan.md`):
  1. Replace the simple LLM call in `main.py` with `agent_graph.invoke(state)`.
  2. Map the FastAPI request/response to the `AgentState` schema.
  3. Return the `agent_response` produced by the graph.
- Until that change is merged, you can still run the graph manually:

```
python backend/test_langgraph_brain.py
```

→ Shows the state transitions and calculated premium.

---

7. Diagram – Action Flow (Mermaid)

```
flowchart TD
    UI[User UI] -->|POST /api/chat| API[FastAPI /api/chat]
    API -->|store message| Session[In-memory session dict]
    API -->|RAG? (keywords)| RAG[RAG search]
    RAG -->|context| API
    API -->|LLM call| LLM[Gemini LLM]
    LLM -->|response| API
    API -->|return| UI
    click API "http://127.0.0.1:8001/api/chat" "API endpoint"
    click RAG "backend/rag_system.py" "RAG implementation"
    click LLM "https://ai.google.dev/" "Gemini LLM"
```

*When LangGraph is integrated, the **LLM call** box will be replaced by `agent_graph.invoke(state)`, and the graph will manage the RAG, premium calculation, and response generation.*

8. Google Search as a Tool

- The current code **does not expose** a generic Google Search tool. Only the **RAG** system searches the local knowledge base.
- If you want a searchable web-tool, you can add a new LangChain tool that calls the Google Custom Search API and register it in `langgraph_agent.py`.

9. Deploying to Google Cloud Run

1. **Build Docker images** for the backend (and optionally the frontend).

```
# Backend Dockerfile (backend/Dockerfile)
FROM python:3.12-slim
WORKDIR /app
COPY backend/requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY backend/ .
EXPOSE 8080
ENV PORT=8080
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

```
# Build and push backend image
cd backend
PROJECT_ID=$(gcloud config get-value project)
IMAGE_BACKEND=gcr.io/$PROJECT_ID/insurance-agent-backend
docker build -t $IMAGE_BACKEND .
docker push $IMAGE_BACKEND
```

2. **Deploy the backend to Cloud Run** (allow unauthenticated access):

```
gcloud run deploy insurance-agent-backend \
  --image $IMAGE_BACKEND \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars GEMINI_API_KEY=$GEMINI_API_KEY
```

   Note the service URL returned, e.g. `https://insurance-agent-backend-xxxxx-uc.a.run.app`.
3. **Frontend (optional)** – build a static image or serve via Cloud Storage. Example using a Docker image:

```
# Frontend Dockerfile (frontend/Dockerfile)
FROM node:20-alpine AS builder
WORKDIR /app
```

```
COPY frontend/package*.json ./
RUN npm ci
COPY frontend/ ./
RUN npm run build
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 8080
CMD ["nginx", "-g", "daemon off;"]
```

```
# Build and push frontend image
cd frontend
IMAGE_FRONTEND=gcr.io/$PROJECT_ID/insurance-agent-frontend
docker build -t $IMAGE_FRONTEND .
docker push $IMAGE_FRONTEND
```

4. **Deploy the frontend to Cloud Run**, passing the backend URL as an environment variable:

```
BACKEND_URL=$(gcloud run services describe insurance-agent-backend \
  --platform managed --region us-central1 \
  --format='value(status.url)')
gcloud run deploy insurance-agent-frontend \
  --image $IMAGE_FRONTEND \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars REACT_APP_BACKEND_URL=$BACKEND_URL
```

5. **Verify** the deployment:
   - Open the frontend URL returned by Cloud Run.
   - Interact with the chatbot; it will call the backend URL you supplied.
   - Check the backend health endpoint: `curl https://<backend-url>/health`.
   - Use Cloud Run logs (`gcloud logs read`) to inspect session data if needed.

---

10. Checklist for the Workshop

- [ ] Clone repo & install Python & Node dependencies.
- [ ] Create `.env` with Gemini API key.
- [ ] Run backend (`uvicorn main:app --port 8001`).
- [ ] Run frontend (`npm run dev`).
- [ ] Demonstrate a chat flow (refusal → auto-insurance quote).
- [ ] Show backend session data via `/health`.
- [ ] Run `test_langgraph_brain.py` to illustrate the LangGraph brain.
- [ ] Explain the planned integration (replace LLM call with LangGraph).
- [ ] (Optional) Deploy with Gunicorn + NGINX or Docker.

All instructions are deliberately concise for a 90-minute session but contain the essential details to let participants run, explore, and extend the project on their own.