# 🚀 Workshop Repository Setup Guide

**Complete Implementation Instructions**

---

## 📋 Overview

This guide will help you create a complete, ready-to-use GitHub repository for your workshop participants. They'll be able to clone it and have a working Agentic AI agent in 90 minutes.

---

## 🎯 Repository Structure

```
insurance-agent-workshop/
├── README.md                          # Main documentation
├── .gitignore                         # Git ignore file
├── LICENSE                            # MIT License
│
├── backend/                           # Python backend
│   ├── .env.example                   # Template for API key
│   ├── requirements.txt               # Python dependencies
│   ├── main.py                        # ✅ Already exists
│   ├── langgraph_agent.py            # ✅ Already exists
│   ├── rag_system.py                 # ✅ Already exists
│   ├── document_analyzer.py          # ✅ Already exists
│   ├── system_prompt.py              # NEW - To create
│   ├── tools.py                      # NEW - To create
│   └── test_agent.py                 # NEW - Quick test script
│
├── frontend/                          # React frontend
│   ├── package.json                   # ✅ Already exists
│   ├── vite.config.js                # ✅ Already exists
│   ├── tailwind.config.js            # ✅ Already exists
│   ├── src/
│   │   ├── App.jsx                    # ✅ Already exists
│   │   ├── components/
│   │   │   ├── ChatInterface.jsx      # ✅ Already exists
│   │   │   └── ...                    # Other components
│   │   └── index.css                  # ✅ Already exists
│   └── README.md                      # Frontend setup
│
├── docs/                              # Workshop materials
│   ├── WORKSHOP_90MIN_GUIDE.md        # ✅ Already exists
│   ├── ORCHESTRATOR_QUICK_REFERENCE.md # ✅ Already exists
│   ├── FACILITATOR_NOTES.md           # ✅ Already exists
│   ├── AI_STUDIO_GUIDE.md             # ✅ Already exists
│   ├── VISUAL_LANGGRAPH_GUIDE.md      # ✅ Already exists
│   ├── ARCHITECTURE_OVERVIEW.md       # ✅ Already exists
│   ├── HUMAN_IN_THE_LOOP_GUIDE.md     # ✅ Already exists
│   └── WORKSHOP_QUESTIONNAIRES.md     # ✅ Already exists
```

```
│
└── examples/                        # Example files
    ├── sample_policy.pdf            # Sample insurance doc
    └── test_conversation.txt        # Example chat
```

---

## 📝 Step-by-Step Implementation

### Step 1: Create Missing Backend Files

#### 1.1 Create `backend/system_prompt.py`

```
cd "C:\Users\Naveen Nalajala\.gemini\antigravity\scratch\insurance_agent\backend"
```

Create file with this content:

```
"""
System prompt designed by orchestrators in AI Studio.
This defines the agent's personality and behavior.
"""

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():
    """Get the system prompt for the agent"""
    return INSURANCE_AGENT_PROMPT
```

**1.2 Create `backend/tools.py`**

```
"""
Tools for the insurance agent.
These are functions the agent can call autonomously.
"""

from langchain.tools import tool

@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.

    Args:
        age: Driver's age
        vehicle_year: Year vehicle was manufactured
        years_licensed: Years driver has been licensed
        accidents: Number of accidents in last 3 years
        violations: Number of violations in last 3 years

    Returns:
        dict: Premium breakdown with monthly/annual costs
    """

    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

    # Vehicle age factor
    vehicle_age = 2025 - vehicle_year
    if vehicle_age < 5:
        vehicle_factor = 100  # Newer cars cost more
    elif vehicle_age > 15:
        vehicle_factor = -50  # Older cars cost less
    else:
        vehicle_factor = 0

    annual_premium = (base_rate + age_factor + experience_factor +
                     accident_factor + violation_factor + vehicle_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,
            "vehicle_age_factor": vehicle_factor
        }
    }


@tool
def calculate_home_premium(
    year_built: int,
    square_footage: int,
    construction_type: str,
    dwelling_coverage: int
) -> dict:
    """
    Calculate home insurance premium.

    Args:
        year_built: Year home was built
        square_footage: Total square footage
        construction_type: Type of construction (frame, masonry, etc.)
        dwelling_coverage: Desired dwelling coverage amount

    Returns:
        dict: Premium breakdown
    """

    # Base rate: $0.50 per $1000 of coverage
    base_rate = (dwelling_coverage / 1000) * 0.50

    # Age factor
    home_age = 2025 - year_built
    if home_age < 10:
        age_factor = -50  # Newer homes get discount
    elif home_age > 50:
        age_factor = 200  # Older homes cost more
    else:
        age_factor = 0

    # Size factor
    if square_footage > 3000:
        size_factor = 150
    elif square_footage < 1500:
        size_factor = -50
    else:
        size_factor = 0

    # Construction type factor
    construction_factors = {
        "masonry": -100,  # Brick/stone is safer
        "frame": 0,       # Wood frame is standard
        "mobile": 200     # Mobile homes cost more
    }
    construction_factor = construction_factors.get(construction_type.lower(), 0)

    annual_premium = base_rate + age_factor + size_factor + construction_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,
            "size_adjustment": size_factor,
            "construction_adjustment": construction_factor
        }
    }


# Test the tools
if __name__ == "__main__":
    print("Testing Auto Premium Calculator:")
    auto_result = calculate_auto_premium.invoke({
        "age": 28,
        "vehicle_year": 2020,
        "years_licensed": 10,
        "accidents": 0,
        "violations": 0
    })
    print(f"Monthly: ${auto_result['monthly_premium']}")
    print(f"Annual: ${auto_result['annual_premium']}")
    print(f"Breakdown: {auto_result['breakdown']}")

    print("\nTesting Home Premium Calculator:")
    home_result = calculate_home_premium.invoke({
        "year_built": 2015,
        "square_footage": 2000,
        "construction_type": "frame",
        "dwelling_coverage": 300000
    })
    print(f"Monthly: ${home_result['monthly_premium']}")
    print(f"Annual: ${home_result['annual_premium']}")
    print(f"Breakdown: {home_result['breakdown']}")
```

**1.3 Create `backend/test_agent.py`**

```
"""
Quick test script to verify the agent works.
Run this before the workshop to ensure everything is set up correctly.
"""

import os
from dotenv import load_dotenv

load_dotenv()

def test_imports():
    """Test that all required packages are installed"""
    print("Testing imports...")
    try:
        import langchain
        print("✅ langchain")
    except ImportError:
        print("❌ langchain - run: pip install langchain")
        return False

    try:
        import langgraph
        print("✅ langgraph")
    except ImportError:
        print("❌ langgraph - run: pip install langgraph")
        return False

    try:
        from langchain_google_genai import ChatGoogleGenerativeAI
        print("✅ langchain-google-genai")
    except ImportError:
        print("❌ langchain-google-genai - run: pip install langchain-google-
genai")
```

```
        return False

    try:
        import chromadb
        print("✅ chromadb")
    except ImportError:
        print("❌ chromadb - run: pip install chromadb")
        return False

    return True


def test_api_key():
    """Test that Gemini API key is set"""
    print("\nTesting API key...")
    api_key = os.getenv("GEMINI_API_KEY")

    if not api_key:
        print("❌ GEMINI_API_KEY not found in .env file")
        print("   Create a .env file with: GEMINI_API_KEY=your_key_here")
        return False

    if not api_key.startswith("AIza"):
        print("⚠️  API key doesn't start with 'AIza' - might be invalid")
        return False

    print(f"✅ API key found: {api_key[:10]}...")
    return True


def test_llm():
    """Test that Gemini LLM works"""
    print("\nTesting Gemini LLM...")
    try:
        from langchain_google_genai import ChatGoogleGenerativeAI
        from langchain.schema import HumanMessage

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

        response = llm.invoke([HumanMessage(content="Say 'Hello Workshop!'")])
        print(f"✅ LLM response: {response.content}")
        return True
    except Exception as e:
        print(f"❌ LLM test failed: {e}")
        return False


def test_tools():
    """Test that tools work"""
    print("\nTesting tools...")
    try:
        from tools import calculate_auto_premium

        result = calculate_auto_premium.invoke({
            "age": 28,
            "vehicle_year": 2020,
            "years_licensed": 10,
            "accidents": 0,
            "violations": 0
        })

        print(f"✅ Tool result: ${result['monthly_premium']}/month")
        return True
    except Exception as e:
        print(f"❌ Tool test failed: {e}")
        return False


def test_rag():
    """Test that RAG system works"""
    print("\nTesting RAG system...")
    try:
        from rag_system import search_knowledge

        results = search_knowledge("What is collision coverage?", k=1)
        if results:
            print(f"✅ RAG search returned {len(results)} results")
            print(f"   Sample: {results[0].page_content[:100]}...")
            return True
        else:
            print("⚠️  RAG search returned no results")
            return False
    except Exception as e:
        print(f"❌ RAG test failed: {e}")
        return False


def main():
    """Run all tests"""
    print("="*60)
    print("🧪 Testing Workshop Setup")
    print("="*60)

    tests = [
        ("Imports", test_imports),
        ("API Key", test_api_key),
        ("Gemini LLM", test_llm),
        ("Tools", test_tools),
        ("RAG System", test_rag)
    ]

    results = {}
    for name, test_func in tests:
        try:
            results[name] = test_func()
        except Exception as e:
            print(f"❌ {name} test crashed: {e}")
            results[name] = False

    print("\n" + "="*60)
    print("📊 Test Results")
    print("="*60)

    for name, passed in results.items():
        status = "✅ PASS" if passed else "❌ FAIL"
        print(f"{status} - {name}")

    all_passed = all(results.values())

    if all_passed:
        print("\n🎉 All tests passed! Ready for workshop!")
    else:
        print("\n⚠️  Some tests failed. Fix issues before workshop.")

    return all_passed


if __name__ == "__main__":
    success = main()
    exit(0 if success else 1)
```

---

### Step 2: Create Configuration Files

#### 2.1 Create `backend/.env.example`

```
# Copy your .env to .env.example (without the actual key)
# Then edit .env.example to have placeholder
```

Content:

```
# Google Gemini API Key
# Get yours at: https://aistudio.google.com/
GEMINI_API_KEY=your_api_key_here
```

#### 2.2 Create `.gitignore`

```
# Python
__pycache__/
*.py[cod]
*$py.class
```

```
*.so
.Python
venv/
env/
.env
*.egg-info/
dist/
build/

# Node
node_modules/
npm-debug.log*
.pnpm-debug.log*

# IDEs
.vscode/
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db

# Project specific
backend/insurance_knowledge_db/
*.db
*.sqlite
agent_graph.png

# Logs
*.log
```

### 2.3 Create `LICENSE`

```
MIT License

Copyright (c) 2025 [Your Name]

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

---

### Step 3: Create Main README.md

Create `README.md` in the root directory with comprehensive documentation (I'll create this as a separate file).

---

### Step 4: Test Everything

```
# Navigate to backend
cd backend

# Activate virtual environment
venv\Scripts\activate

# Run test script
python test_agent.py
```

**Expected output**:

```
🧪 Testing Workshop Setup
============================================================
Testing imports...
✅ langchain
✅ langgraph
✅ langchain-google-genai
✅ chromadb

Testing API key...
✅ API key found: AIzaSyAkWC...

Testing Gemini LLM...
✅ LLM response: Hello Workshop!

Testing tools...
✅ Tool result: $75.0/month

Testing RAG system...
✅ RAG search returned 1 results

============================================================
📊 Test Results
============================================================
✅ PASS - Imports
✅ PASS - API Key
✅ PASS - Gemini LLM
✅ PASS - Tools
✅ PASS - RAG System

🎉 All tests passed! Ready for workshop!
```

---

### Step 5: Initialize Git Repository

```
# Navigate to project root
cd "C:\Users\Naveen Nalajala\.gemini\antigravity\scratch\insurance_agent"

# Initialize git
git init

# Add all files
git add .

# Create first commit
git commit -m "Initial commit: Complete Agentic AI workshop code"

# Create GitHub repository (do this on GitHub.com)
# Then connect it:
git remote add origin https://github.com/YOUR_USERNAME/insurance-agent-
workshop.git
git branch -M main
git push -u origin main
```

---

### Step 6: Create GitHub Repository

1. **Go to**: https://github.com/new
2. **Repository name**: `insurance-agent-workshop`
3. **Description**: "Build Enterprise AI Agents with Google - 90-Minute Workshop"
4. **Public** repository
5. **Don't** initialize with README (we already have one)
6. **Click** "Create repository"
7. **Follow** the instructions to push your existing repository

---

### Step 7: Add Repository Topics

On GitHub, add these topics:

- `agentic-ai`
- `langgraph`
- `langchain`
- `google-gemini`

- `rag`
- `workshop`
- `ai-education`
- `insurance`

---

### Step 8: Create Release

1. **Go to**: Releases → Create a new release
2. **Tag**: `v1.0.0`
3. **Title**: "Workshop Ready - v1.0.0"
4. **Description**:

```
🎉 Complete workshop code for building Agentic AI systems!

What's included:
✅ Full LangGraph agent implementation
✅ RAG system with Chroma
✅ Gemini integration
✅ React frontend
✅ Complete workshop materials
✅ Step-by-step guides

Ready for 40 participants!
```

5. **Publish release**

---

## ✅ Final Checklist

Before workshop day:

- ☐ All backend files created (`system_prompt.py`, `tools.py`, `test_agent.py`)
- ☐ Configuration files in place (`.env.example`, `.gitignore`, `LICENSE`)
- ☐ Main README.md created
- ☐ Test script passes all tests
- ☐ Git repository initialized
- ☐ Pushed to GitHub
- ☐ Repository is public
- ☐ Topics added
- ☐ Release created
- ☐ Tested cloning and setup from scratch

---

## 📦 Participant Setup Instructions

**What participants will do**:

```
# 1. Clone repository
git clone https://github.com/YOUR_USERNAME/insurance-agent-workshop.git
cd insurance-agent-workshop

# 2. Backend setup
cd backend
python -m venv venv
venv\Scripts\activate  # Windows
pip install -r requirements.txt

# 3. Create .env file
copy .env.example .env
# Edit .env and add your GEMINI_API_KEY

# 4. Test setup
python test_agent.py

# 5. Run backend
python main.py

# 6. Frontend setup (new terminal)
cd frontend
npm install
npm run dev
```

---

## 🎯 Next Steps

1. **Create the missing files** (system_prompt.py, tools.py, test_agent.py)
2. **Run test_agent.py** to verify everything works
3. **Create main README.md** (I'll provide template)
4. **Initialize Git and push to GitHub**
5. **Test by cloning fresh copy** and following setup instructions

---

**Ready to implement? Let me know if you want me to create any of these files for you!**