🏦 ADK Finance Assistance Agent

A beginner-friendly guide to building an AI-powered Finance Assistant using Google's Agent Development Kit (ADK).

📚 Table of Contents

🎯 Introduction

This project demonstrates how to build an intelligent Finance Assistant Agent using Google's Agent Development Kit (ADK). The agent can answer finance-related questions, analyze personal finance details, and provide investment planning advice by searching for real-time information.

What Makes This Project Special?

- **Progressive Learning**: Starts simple and gradually adds complexity
- **Real-World Application**: Solves actual finance planning problems
- **Multi-Agent Architecture**: Demonstrates how agents can work together
- **Live Data Integration**: Uses Google Search to fetch current interest rates and market data

---

📖 What You'll Learn

By following this tutorial, you'll learn how to:

1. Create a basic LLM agent that answers questions
2. Add custom tools to enhance agent capabilities
3. Build a multi-agent system where agents collaborate
4. Integrate Google Search for real-time information
5. Structure an ADK project properly

✅ Prerequisites

Before starting, make sure you have:

- **Python 3.8+** installed on your system
- **Google API Key** for accessing Gemini models (Get it here)
- Basic understanding of Python
- Terminal/Command Prompt knowledge

🚀 Setup

1. Create Python Virtual Environment

For Mac OS / Linux:

```
# Navigate to your project folder
cd /path/to/your/project

# Create virtual environment
python3 -m venv .venv

# Activate virtual environment
source .venv/bin/activate
```

For Windows:

```
# Navigate to your project folder
cd C:\path\to\your\project

# Create virtual environment
python -m venv .venv

# Activate virtual environment
.venv\Scripts\activate.bat
```

2. Install Dependencies

Create a `requirements.txt` file with the following content:

```
google-adk
python-dotenv
```

Install the dependencies:

```
pip3 install -r requirements.txt
```

3. Set Up API Key

Create a `.env` file in your project root:

```
GOOGLE_API_KEY=your_api_key_here
```

> **Important**: Replace `your_api_key_here` with your actual Google API key.

📝 Step-by-Step Tutorial

We'll build this project in three progressive steps, each adding new capabilities to our finance assistant.

Step 1: Basic Finance Assistant (No Tools)

In this first step, we'll create a simple finance assistant that can answer generic finance questions without any external tools.

📁 Create the Project Structure

```
ADK_Finance_Assistance_Agent/
├── finance_assistant_agent/
│   ├── __init__.py
│   └── agent.py
└── requirements.txt
```

💻 Code Implementation

File: `finance_assistant_agent/__init__.py`

```python
from . import agent
```

File: `finance_assistant_agent/agent.py`

```python
from google.adk.agents import LlmAgent

finance_assistance_agent = LlmAgent(
    name="finance_assistance_agent",
    model="gemini-2.5-flash",
    description="A simple finance assistant that helps with user's finance
goals.",
    instruction="""You are a friendly finance assistant.
        You can help answer user's generic questions on finance and help
plan
```

```
        their finance goals. Be more friendly and positive.
    """,
    tools=[]  # No tools yet!
)

root_agent = finance_assistance_agent
```

🔍 Understanding the Code

Let's break down what each part does:

1. `LlmAgent`: This is the core class from Google ADK that creates an AI agent powered by a Large Language Model.
2. `name`: A unique identifier for your agent. This helps when you have multiple agents.
3. `model`: Specifies which Gemini model to use. We're using `gemini-2.5-flash` which is fast and efficient.
4. `description`: A brief explanation of what this agent does. This is useful when other agents need to call this agent.
5. `instruction`: Detailed instructions that guide the agent's behavior. Think of this as the agent's "personality" and "job description."
6. `tools=[]`: An empty list because we haven't added any tools yet.
7. `root_agent`: ADK looks for this variable to know which agent to run first.

🧪 Testing Step 1

Run the agent using ADK's web interface:

```
adk web
```

Open your browser and navigate to the provided URL (usually `http://localhost:8000`).

Example Interaction:

User: "Explain Recurring Deposit"

Agent Response:

> "A Recurring Deposit (RD) is a type of term deposit offered by banks and financial institutions. It allows you to deposit a fixed amount of money regularly (monthly, quarterly, etc.) for a predetermined period. At the end of the tenure, you receive the maturity amount which includes your principal deposits plus the interest earned.
>
> Key Features:

> - Fixed monthly installments
> - Higher interest rates than savings accounts
> - Flexible tenure options (6 months to 10 years)
> - Good for building savings discipline
> - Premature withdrawal options available with penalties"

✅ **Step 1 Complete!** Your agent can now answer general finance questions using its built-in knowledge.

Step 2: Adding Custom Tools

Now let's enhance our agent by adding a custom tool that provides the user's personal finance details. This allows the agent to give personalized advice.

💻 **Updated Code**

File: `finance_assistant_agent/agent.py`

```
from google.adk.agents import LlmAgent
from typing import Dict

def get_user_personal_finance_details() -> Dict:
    """
    Gets users personal finance details like salary, expense and savings
capacity.
    """
    return {
        "salary": 50000,
        "expense": {
            "EMI_Expense": 25000,
            "Essentials": 5000,
            "Entertainment": 5000,
            "Shopping and Travel": 5000
        },
        "savings": 10000
    }

finance_assistance_agent = LlmAgent(
    name="finance_assistance_agent",
    model="gemini-2.5-flash",
    description="A simple finance assistant that helps with user's finance
goals.",
    instruction="""You are a friendly finance assistant.
        You can help answer user's generic questions on finance and help
plan
        their finance goals. Be more friendly and positive.

        You have a tool called get_user_personal_finance_details which
will
        give you the user's current finance details.
    """,
    tools=[get_user_personal_finance_details]  # Tool added!
)

root_agent = finance_assistance_agent
```

🔍 Understanding the Custom Tool

1. **Function Definition**: We create a Python function `get_user_personal_finance_details()` that returns a dictionary with the user's financial information.

2. **Type Hints**: `-> Dict` indicates this function returns a dictionary. This helps ADK understand the tool better.

3. **Docstring**: The triple-quoted string under the function name is crucial! ADK uses this to understand when and how to use the tool.

4. **Return Value**: We return a structured dictionary with:

    - `salary`: Monthly income (₹50,000)
    - `expense`: Breakdown of different expense categories
    - `savings`: Current monthly savings (₹10,000)

5. **Tools List**: We pass the function to the `tools` parameter. ADK automatically converts it into a tool the agent can use.

🧪 Testing Step 2

Restart the ADK web interface:

```
adk web
```

Example Interaction:

User: "Can you analyse my financial details and suggest how to save 15000 per month"

Agent Response:

> "Let me check your current financial situation...
>
> Based on your details:
>
> - **Monthly Salary**: ₹50,000
> - **Current Expenses**: ₹40,000
>   - EMI: ₹25,000
>   - Essentials: ₹5,000
>   - Entertainment: ₹5,000
>   - Shopping & Travel: ₹5,000
> - **Current Savings**: ₹10,000
>
> To save ₹15,000 per month (an additional ₹5,000), here are my suggestions:

> 1. **Reduce Entertainment Spending**: Cut down from ₹5,000 to ₹3,000 (Save ₹2,000)
> 2. **Optimize Shopping & Travel**: Reduce from ₹5,000 to ₹3,000 (Save ₹2,000)
> 3. **Find Additional Income**: Consider a side hustle for extra ₹1,000
>
> This way, you can achieve your goal of saving ₹15,000 monthly while maintaining essential expenses!"

✅ **Step 2 Complete!** Your agent can now access user data and provide personalized financial advice.

Step 3: Multi-Agent Setup

In this final step, we'll add a second agent that can search the internet for real-time information like current FD interest rates. The two agents will work together to provide comprehensive financial planning.

📁 Updated Project Structure

```
ADK_Finance_Assistance_Agent/
├── finance_assistant_agent/
│   ├── __init__.py
│   └── agent.py
├── investment_plan_agent/
│   ├── __Init__.py
│   └── agent.py
└── requirements.txt
```

💻 Code Implementation

File: `investment_plan_agent/__Init__.py`

```python
from . import agent
```

File: `investment_plan_agent/agent.py`

```python
from google.adk.agents import LlmAgent
from google.adk.tools import google_search

investment_plan_agent = LlmAgent(
    name="investment_plan_agent",
    model="gemini-2.5-flash",
    description=" An investment plan assistant who can use Google Search to find latest information " \
    "and assist users in creating a savings plan",
    instruction="""You are a friendly finance assistant.
        You can help analyse user's monthly spending and find out ways to
        reduce spending and increase savings to achieve their goal.
        ALWAYS use the google_search tool when asked about:
        - Stock prices (e.g., "Tesla stock price", "TSLA latest price")
        - Market data, financial news, or company information
        - ANY question containing words like "latest", "current", "today",
"now", "recent"

        After searching, provide the factual data from the search results
with specific numbers and sources.
    """,
    tools=[google_search]
)
```

File: `finance_assistant_agent/agent.py` (Updated)

```
from google.adk.agents import LlmAgent
from typing import Dict
from google.adk.tools.agent_tool import AgentTool
from investment_plan_agent.agent import investment_plan_agent

def get_user_personal_finance_details() -> Dict:
    """
    Gets users personal finance details like salary, expense and savings
capacity.
    """
    return {
        "salary": 50000,
        "expense": {
            "EMI_Expense": 25000,
            "Essentials": 5000,
            "Entertainment": 5000,
            "Shopping and Travel": 5000
        },
        "savings": 10000
    }

finance_assistance_agent = LlmAgent(
    name="finance_assistance_agent",
    model="gemini-2.5-flash",
    description="A simple finance assistant that helps with user's finance
goals.",
    instruction="""You are a friendly finance assistant.
        You can help answer user's generic questions on finance and help
plan
        their finance goals. Be more friendly and positive.

        You have two tools to use to complete your task.
        1. get_user_personal_finance_details - This tool will give you the
user's current finance details
        2. investment_plan_agent - This tool can perform Google Search to
get any
        latest information from websites and will be able to ask more
details
```

```
        from the user and plan their savings goal.

        ALWAYS use the investment_plan_agent with google_search tool when
asked about:
        - Stock prices (e.g., "Tesla stock price", "TSLA latest price")
        - Market data, financial news, or company information
        - ANY question containing words like "latest", "current", "today",
"now", "recent"
    """,
    tools=[AgentTool(investment_plan_agent),
get_user_personal_finance_details]
)

root_agent = finance_assistance_agent
```

🔍 Understanding Multi-Agent Architecture

Let's understand how the agents work together:

1. **Investment Plan Agent**:
   - Specialized agent focused on investment planning
   - Has access to `google_search` tool for real-time data
   - Can search for current FD rates, stock prices, etc.
2. **AgentTool Wrapper**:

   ```
   AgentTool(investment_plan_agent)
   ```

   This wraps another agent as a tool, allowing the main agent to delegate tasks to it.
3. **Agent Collaboration**:
   - Main agent (`finance_assistance_agent`) receives the user query
   - It uses `get_user_personal_finance_details()` to get personal data
   - It calls `investment_plan_agent` (via `AgentTool`) for market research
   - Investment agent uses `google_search` to find current FD rates
   - Results flow back to the main agent
   - Main agent combines all information and responds to the user
4. **Clear Instructions**: Both agents have clear instructions about when to use their tools, preventing confusion and ensuring efficient task delegation.

🧪 Testing Step 3

Restart ADK web interface:

```
adk web
```

Example Interaction:

User: "Can you analyse my financial details and suggest how to save 15000 per month and also check the current FD interest rates for SBI and tell me if I invest this 15000 Rupees in FD every month can I achieve 1000000 in 3 years?"

Agent Response:

> "Let me analyze your situation comprehensively...
>
> **Your Current Financial Situation:**
>
> - Monthly Salary: ₹50,000
> - Current Savings: ₹10,000
> - Total Expenses: ₹40,000
>
> **Expense Breakdown:**
>
> - EMI: ₹25,000
> - Essentials: ₹5,000
> - Entertainment: ₹5,000
> - Shopping & Travel: ₹5,000
>
> **To Save ₹15,000 per month:**
>
> 1. Reduce Entertainment: ₹5,000 → ₹3,000 (Save ₹2,000)
> 2. Cut Shopping & Travel: ₹5,000 → ₹3,000 (Save ₹2,000)
> 3. Optimize Essentials: ₹5,000 → ₹4,000 (Save ₹1,000)
>
> **Current SBI FD Rates** (as per latest search):
>
> - For 3-year tenure: Approximately 6.5% - 7.0% per annum
> - Senior Citizens: 7.0% - 7.5% per annum
>
> **Investment Calculation:** If you invest ₹15,000 every month in an SBI FD at 7% annual interest:
>
> - Monthly Investment: ₹15,000
> - Duration: 36 months (3 years)
> - Total Principal: ₹5,40,000
> - Estimated Interest: ~₹60,000 - ₹70,000
> - **Maturity Amount: ~₹6,00,000 - ₹6,10,000**
>
> **Conclusion:** Unfortunately, with ₹15,000 monthly investment in FD at current rates, you'll reach approximately ₹6 lakhs in 3 years, which is short of your ₹10 lakh goal.
>
> **Alternative Suggestions:**
>
> 1. Increase monthly investment to ₹25,000 if possible
> 2. Consider higher-return options like mutual funds (market-linked)
> 3. Extend the investment period to 5-6 years
> 4. Combine FD with equity investments for better returns"

✅ **Step 3 Complete!** You now have a multi-agent system that provides comprehensive financial planning with real-time market data!

📂 Project Structure

Your final project structure should look like this:

```
ADK_Finance_Assistance_Agent/
│
├── finance_assistant_agent/
│   ├── __init__.py         # Module initializer
│   └── agent.py             # Main finance assistant agent
│
├── investment_plan_agent/
│   ├── __Init__.py          # Module initializer
│   └── agent.py             # Investment planning agent with Google
Search
│
├── requirements.txt         # Project dependencies
├── .env                     # API keys (not in version control)
└── README.md                # This file
```

🎮 Running the Application

Method 1: ADK Web Interface (Recommended for Beginners)

```
# Activate virtual environment (if not already activated)
source .venv/bin/activate  # Mac/Linux
# or
.venv\Scripts\activate.bat  # Windows

# Run ADK web interface
adk web
```

This will start a local web server (usually at `http://localhost:8000`) where you can interact with your agent through a chat interface.

Method 2: Command Line

```
# Run in terminal mode
adk run
```

This provides a command-line interface to interact with your agent.

🎓 Key Concepts

1. **LLM Agent**

An AI agent powered by a Large Language Model (like Gemini) that can understand natural language, make decisions, and use tools to accomplish tasks.

2. **Tools**

Functions or capabilities that an agent can use to perform specific tasks:

- **Custom Functions**: Python functions you write (like `get_user_personal_finance_details`)
- **Built-in Tools**: Provided by ADK (like `google_search`)
- **Agent Tools**: Other agents wrapped as tools for delegation

3. **Multi-Agent System**

Multiple specialized agents working together, each handling specific aspects of a larger task. Benefits include:

- Better separation of concerns
- Easier to maintain and debug
- Ability to specialize agents for specific tasks
- Scalable architecture

4. **Instructions**

Clear directions given to agents about their role, capabilities, and when to use specific tools. Good instructions lead to better agent performance.

5. **Root Agent**

The entry point of your application. ADK looks for the `root_agent` variable to know which agent to initialize first.

🎯 What You've Learned

Congratulations! You've successfully built a complete AI-powered finance assistant. Here's what you've accomplished:

✅ Created a basic LLM agent
✅ Added custom tools for personalized responses
✅ Built a multi-agent system with specialized agents
✅ Integrated real-time data using Google Search
✅ Structured a professional ADK project

🚀 Next Steps

Want to expand this project? Here are some ideas:

1. **Add More Agents**: Create agents for tax planning, insurance advice, etc.
2. **Database Integration**: Store user finance data in a database instead of hardcoded values
3. **More Custom Tools**: Add tools for expense tracking, budget creation, etc.
4. **Authentication**: Add user authentication to handle multiple users
5. **Visualization**: Create charts and graphs for financial data
6. **Notifications**: Set up alerts for savings goals and investment opportunities

---

📚 Resources

- Google ADK Documentation
- Gemini API Documentation
- Python Virtual Environments

📺 Video Tutorial

Watch the full video tutorial on YouTube: https://youtu.be/OA6y5JoEjGs

---

💬 Support

If you have questions or need help:

- Comment on the YouTube video
- Check the Google ADK documentation
- Join the community discussions

📄 License

Happy Coding! 🎉

Built with ❤️ using Google's Agent Development Kit (ADK)