Smart Inventory Agent: Autonomous Supply Chain Assistant (Updated)

This project introduces a fully agentic AI system designed to streamline inventory management and procurement. Built with LangChain, SQLite, and SMTP, the Smart Inventory Agent autonomously monitors stock levels, identifies low-stock items, searches for suppliers, and sends email summaries with actionable insights.

Recommended Path: Build It Yourself

To truly understand how this agent works—and to gain hands-on experience with LangChain, tool orchestration, and autonomous workflows—I strongly encourage you to follow the step-by-step project manual and build it from scratch. You'll learn how each module fits together, how the agent reasons across tasks, and how to customize it for your own use cases.

⚡ *Shortcut: Clone the Completed Project*

If you'd prefer to skip the build process and dive straight into a working version, you can clone the full project into your desired directory:

```
git clone --single-branch --branch step-4 https://github.com/solomontessema/smart-inventory-agent.git
```

This will give you immediate access to the full codebase, including the agent logic, tools, database schema, and web interface.

▶️ Activate the Environment

- **Windows**:

```
venv\Scripts\activate
```

- **macOS/Linux**:

```
source venv/bin/activate
```

Once activated, your terminal will show the environment name (e.g., `(venv)`), and all `pip` installs will go into this isolated space.

📦 *Install Dependencies*

```
pip install -r requirements.txt
```

📂 *Git Initialization*

To enable version control and streamline collaboration, initialize Git in the root of your project:

```
git init
```

✅ *First Commit*

```
git add .
git commit -m "Initial project structure for Smart Inventory Agent"
```

🗓️ *Optional: Create a GitHub Repo*

```
git remote add origin https://github.com/yourusername/smart-inventory-agent.git
git push -u origin main
```

This keeps your main branch clean while you builand

✅*Add your API Keys, Email Address, and Email app password to the .env file*

```
OPENAI_API_KEY=your open api key
TAVILY_API_KEY=your tavily api key
AGENT_NAME=Inventory Agent
AGENT_EMAIL_ADDRESS=agent_email_address@example.com
AGENT_EMAIL_PASSWORD=agent email app password
BOSS_NAME=boss_first_name
BOSS_EMAIL_ADDRESS = boss_email@example.com
```

Run main.py and chat.py and see if they are working as expected. (Those two are our entry points.)

Once everything runs smoothly, Create a batch file (run_inventory_agent.bat) prefereably outside the project folder and add the following and run it by double clicking the batch file.

```
@echo off
REM Run Smart Inventory Agent
cd C:\...path to your project folder\smart-inventory-agent-final
call venv\Scripts\activate.bat
python main.py
```

Use windows Task Scheduler to Schedule the batch file run everyday.

Recommended Path: Build It Yourself

Project Concept

Smart Inventory Agent: A LangChain-powered agent that monitors inventory levels, searches suppliers, and sends purchase suggestions via email. It uses SQLite for inventory data, and SMTP to communicate with the user. The agent tracks its actions in a log and supports natural-language interaction through a Flask web interface.

📁 Project Structure

```
smart-inventory-agent/
├── agents/
│   └── inventory_agent.py         #Agent that checks inventory, search supply and send email
│
│
├── database/
│   └── data.db                    # Note: SQLite database storing inventory data
│
│
├── tools/
│   └── database_reader.py         # Tool to query inventory database
│   └── db_connector.py            # A class for use by database_reader tool
│   └── email_sender.py            # Tool to send emails via SMTP
│   └── log_tracker.py             # Tracks agent actions and stores execution logs
│   └── web_searcher.py            # Tool to search suppliers
│
│
├── web_app.py                     # (Optional) Flask web app to interact with the agent
│
├── templates/
│       └── index.html             # Web interface for use by the web_app.py
│
│
├── .env                           # API Keys and other secrets
├── .gitignore                     # list of files and directories not to include in git.
├── config.py                      # Configuration for thresholds, email settings, etc.
├── main.py                        # Entry point to run the agent
├── README.md                      # Project overview and setup instructions
└── requirements.txt               # Python dependencies
```

Open a command prompt in the location where you'd like to create the project directory, then copy and paste the following CMD script to initialize the structure.

```
@echo off
REM Setup script for smart-inventory-agent project

REM Create root directory
mkdir smart-inventory-agent
cd smart-inventory-agent

REM Create subdirectories
mkdir agents
mkdir tools
mkdir database
mkdir templates
```

```
REM Create placeholder files
echo # LangChain agent that checks inventory and triggers supplier search > agents\inventory_agent.py

echo # Tool to query inventory database > tools\database_reader.py
echo # DB Connector Class > tools\db_connector.py
echo # Tool to search the web > tools\web_search_tool.py
echo # Tool to send emails via SMTP > tools\email_sender.py
echo # Tracks agent actions and stores execution logs > tools\log_tracker.py

echo # Flask web app to interact with the agent > web_app.py
echo # Web interface for chat and status > templates\index.html

echo # Configuration for thresholds, email settings, etc. > config.py
echo # Entry point to run the agent > main.py
echo # Python dependencies > requirements.txt
echo # Project overview and setup instructions > README.md
echo # API Keys and other secrets > .env
echo # gitignore > .gitignore

echo Project directories and files created successfully.
pause
```

🗄️ Creating a SQLite Database and Tables

Start a command prompt In the project's database directory and run sqlite3 data.db to initialize a new SQLite database named data.db.

```
sqlite3 data.db;
```

Create products table

```
CREATE TABLE products (
    barcode TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    threshold INTEGER NOT NULL
);
```

Insert data into products table

```
INSERT INTO products (name, barcode, threshold) VALUES
('Gillette Mach3 Razor Blades (8 ct)', '047400245909', 80),
('Kellogg's Corn Flakes 18 oz', '038000001019', 90),
('Coca-Cola Classic 12 oz Can', '049000028911', 120),
('Tylenol Extra Strength 100 ct', '300450444305', 40),
('Colgate Total Toothpaste 4.8 oz', '035000521019', 50);
```

Create the inventory table

```
CREATE TABLE inventory (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    barcode TEXT NOT NULL,
```

```
    quantity INTEGER NOT NULL,
    transaction_date TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (barcode) REFERENCES products(barcode)
);
```

Insert data into inventory table.

```
 INSERT INTO inventory (name, barcode, quantity, transaction_date) VALUES
('Gillette Mach3 Razor Blades (8 ct)', '047400245909', 90, '2025-10-01 07:45:00'),
('Kellogg's Corn Flakes 18 oz', '038000001019', 120, '2025-10-01 08:30:00'),
('Coca-Cola Classic 12 oz Can', '049000028911', 100, '2025-10-01 09:00:00'),
('Tylenol Extra Strength 100 ct', '300450444305', 70, '2025-10-01 10:00:00'),
('Colgate Total Toothpaste 4.8 oz', '035000521019', 60, '2025-10-01 11:00:00'),

('Kellogg's Corn Flakes 18 oz', '038000001019', -25, '2025-10-02 12:00:00'),
('Colgate Total Toothpaste 4.8 oz', '035000521019', -10, '2025-10-02 13:00:00'),
('Coca-Cola Classic 12 oz Can', '049000028911', -20, '2025-10-02 14:30:00'),
('Tylenol Extra Strength 100 ct', '300450444305', -15, '2025-10-02 15:00:00'),
('Gillette Mach3 Razor Blades (8 ct)', '047400245909', -10, '2025-10-02 16:00:00'),

('Kellogg's Corn Flakes 18 oz', '038000001019', -35, '2025-10-03 14:00:00'),
('Colgate Total Toothpaste 4.8 oz', '035000521019', -15, '2025-10-03 15:00:00'),
('Coca-Cola Classic 12 oz Can', '049000028911', -30, '2025-10-03 16:45:00'),
('Tylenol Extra Strength 100 ct', '300450444305', -20, '2025-10-03 17:00:00'),
('Gillette Mach3 Razor Blades (8 ct)', '047400245909', -15, '2025-10-03 18:00:00'),

('Kellogg's Corn Flakes 18 oz', '038000001019', 60, '2025-10-04 09:00:00'),
('Coca-Cola Classic 12 oz Can', '049000028911', 50, '2025-10-04 10:15:00'),
('Tylenol Extra Strength 100 ct', '300450444305', 40, '2025-10-04 11:00:00'),
('Gillette Mach3 Razor Blades (8 ct)', '047400245909', 30, '2025-10-04 12:00:00'),
('Colgate Total Toothpaste 4.8 oz', '035000521019', 30, '2025-10-04 17:00:00');
```

Create logs table

```
 CREATE TABLE logs (
    id INTEGER PRIMARY KEY,
    timestamp TEXT,
    action TEXT,
    details TEXT
);
```

Exit Sqlite

✅ Add your API Keys, Email Address, and Email app password to the .env file

```
OPENAI_API_KEY=your open api key
TAVILY_API_KEY=your tavily api key
AGENT_NAME=Inventory Agent
AGENT_EMAIL_ADDRESS=agent_email_address@example.com
AGENT_EMAIL_PASSWORD=agent email app password
BOSS_NAME=boss_first_name
BOSS_EMAIL_ADDRESS = boss_email@example.com
```

✅ Create a Virtual Environment

```
python -m venv venv
```

This creates a `venv/` folder containing the Python interpreter and site packages.

▶️ Activate the Environment

- **Windows**:

  ```
  venv\Scripts\activate
  ```

- **macOS/Linux**:

  ```
  source venv/bin/activate
  ```

Once activated, your terminal will show the environment name (e.g., `(venv)`), and all `pip` installs will go into this isolated space.

Add the following to your `requirements.txt` file:

```
langchain==0.3.27
langchain-core==0.3.78
langchain-community==0.3.31
langchain-tavily==0.2.12
langchain-text-splitters==0.3.11
openai==2.6.0
python-dotenv==1.1.1
flask==3.1.2
```

📦 Install Dependencies

```
pip install -r requirements.txt
```

📂 Git Initialization

To enable version control and streamline collaboration, initialize Git in the root of your project:

```
git init
```

📄 Then, configure your `.gitignore` file to exclude sensitive and unnecessary files from version tracking:

```
# .gitignore
```

```
venv/
__pycache__/
*.pyc
*.db
.env
logs/
.vscode/
.idea/
.tox/
.coverage
.cache/
*.log
*.tmp
```

✅ First Commit

```
git add .
git commit -m "Initial project structure for Smart Inventory Agent"
```

🗓️ Optional: Create a GitHub Repo

```
git remote add origin https://github.com/yourusername/smart-inventory-agent.git
git push -u origin main
```

This keeps your main branch clean while you builand

🧠 Agent Implementation

We'll start by creating the core LangChain agents using GPT-4o-mini. Your API key will be loaded from the `.env` file using `python-dotenv`.

✅ Environment Setup

In `config.py`, load your OpenAI key:

```
import os
from dotenv import load_dotenv

load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
TAVILY_API_KEY=os.getenv("TAVILY_API_KEY")
AGENT_NAME=os.getenv("AGENT_NAME")
AGENT_EMAIL_ADDRESS= os.getenv("AGENT_EMAIL_ADDRESS")
AGENT_EMAIL_PASSWORD= os.getenv("AGENT_EMAIL_PASSWORD")
BOSS_NAME=os.getenv("BOSS_NAME")
BOSS_EMAIL_ADDRESS=os.getenv("BOSS_EMAIL_ADDRESS")

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
```

```
DB_PATH = os.path.join(BASE_DIR, "database", "data.db")
```

🧠 Inventory Agent (`agents/inventory_agent.py`)

```
from typing import Any, Dict
from langchain_community.chat_models import ChatOpenAI
from langchain.agents import Tool, AgentExecutor, create_react_agent
from langchain.memory import ConversationBufferMemory
from langchain.prompts import PromptTemplate

from tools.database_reader import read_database_tool
from tools.web_searcher import web_search_tool
from tools.email_sender import send_email_tool
from tools.log_tracker import track_log_tool
from config import OPENAI_API_KEY, AGENT_NAME, BOSS_NAME

# ---------------------------
# LLM
# ---------------------------
llm = ChatOpenAI(
    model="gpt-4o-mini",
    temperature=0,
    api_key=OPENAI_API_KEY
)

# ---------------------------
# Memory
# ---------------------------
memory = ConversationBufferMemory(
    memory_key="chat_history",
    return_messages=True,
    input_key="input"
)

# ---------------------------
# Tools
# ---------------------------
tools = [
    Tool(
        name="Database Reader",
        func=read_database_tool,
        description="Execute a SQL query and return raw tabular results. Use for inventory & thresholds."
    ),
    Tool(
        name="Supplier Finder",
        func=web_search_tool,
        description="Search suppliers by product name, barcode, or category."
    ),
    Tool(
        name="Email Sender",
        func=send_email_tool,
        description="Send an email using: subject || body"
    ),
    Tool(
        name="Log Tracker",
        func=track_log_tool,
```

```
        description="Record actions/results for auditing & debugging."
    ),
]

# ----------------------------
# Prompt
# ----------------------------
prompt_template = PromptTemplate(
    input_variables=["input", "chat_history", "agent_scratchpad", "tools", "tool_names", "AGENT_NAME", "BOSS_NA
    template="""
You are an inventory assistant named {AGENT_NAME}. The user is {BOSS_NAME}.
Task: answer all inventory-related questions and perform actions as requested and log them. To see our database

You have tools:
{tools}

Rules:
- You must NEVER include a Final Answer and an Action in the same response.
- If you have completed the task, first give the Final Answer and STOP.
- In the next step, you may log the action using the Log Tracker tool.
- Prefer ONE SQL that joins products with inventory, aggregates quantity, and filters total < threshold.
- When writing SQL, write ONLY the SQL (no backticks/markdown/comments).
- Always ground your Final Answer in the actual tool Observation.
- If a tool fails, try a corrected attempt; do not conclude after an error.
- Keep the final answer concise and helpful.

When using the Email Sender tool:
- Format the body as professional HTML (use <p>, <ul>, <li>, <b>, <a> tags).
- Do not include Markdown or \n escapes.

Available tool names: {tool_names}

Format:
Thought: reasoning
Action: one of [{tool_names}]
Action Input: input for the action (SQL on one line, no fences)
Observation: (system will insert the tool result)
... (repeat Thought/Action/Action Input/Observation if needed)
Thought: I now know the final answer
Final Answer: concise result grounded in Observation

Thought: I should log this action.
Action: Log Tracker
Action Input: Task Completed | Identified low stock items and emailed summary to boss.

Example (for low stock):
Thought: I should run a single SQL that computes total quantities vs thresholds.
Action: Database Reader
Action Input: SELECT p.name, p.barcode, COALESCE(SUM(i.quantity),0) AS total_quantity, p.threshold
FROM products p LEFT JOIN inventory i ON i.barcode = p.barcode
GROUP BY p.name, p.barcode, p.threshold
HAVING COALESCE(SUM(i.quantity),0) < p.threshold
ORDER BY p.name;

Observation: name|barcode|total_quantity|threshold
Widget A|12345|4|10
Cable|99999|0|5

Thought: I have the products below threshold.
Final Answer: Low stock:
- Widget A (12345): total_quantity=4, threshold=10
```

```
- Cable (99999): total_quantity=0, threshold=5

Thought: I should log this action.
Action: Log Tracker
Action Input: Task Completed | Identified low stock items and emailed summary to boss.




Conversation history:
{chat_history}

User query: {input}

{agent_scratchpad}
"""
)

# ----------------------------
# Agent
# ----------------------------
agent = create_react_agent(llm=llm, tools=tools, prompt=prompt_template)

inventory_agent = AgentExecutor(
    agent=agent,
    tools=tools,
    memory=memory,
    verbose=False,
    handle_parsing_errors=True,
    max_iterations=6,
    return_intermediate_steps=False
)

# ----------------------------
# Runner
# ----------------------------
def run_inventory_agent(user_input: str) -> str:
    result = inventory_agent.invoke({
        "input": user_input,
        "AGENT_NAME": AGENT_NAME,
        "BOSS_NAME": BOSS_NAME
    })

    final_answer = result.get("output", "")
    if final_answer:
        track_log_tool(f"Task Completed | {final_answer[:100]}")
    return final_answer
```

We'll now create the core tools that power the agent. These tools encapsulate the actual logic for querying inventory, searching suppliers, sending emails, and logging actions.

✅ Database Connector (`tools/db_connector.py`)

```
import sqlite3
from config import DB_PATH

class SQLiteConnector:
    def __init__(self, db_path=DB_PATH):
```

```
        uri_path = f"file:///{db_path}"
        self.conn = sqlite3.connect(uri_path, uri=True, check_same_thread=False)
        self.cursor = self.conn.cursor()

    def list_tables(self):
        self.cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
        return [row[0] for row in self.cursor.fetchall()]

    def describe_table(self, table_name):
        self.cursor.execute(f"PRAGMA table_info({table_name});")
        return self.cursor.fetchall()

    def get_schema_summary(self):
        summary = []
        for table in self.list_tables():
            cols = self.describe_table(table)
            formatted = f"Table: {table} Columns: {[col[1] for col in cols]}"
            summary.append(formatted)
        return "\n".join(summary)
```

✅ Inventory Tool (`tools/database_reader.py`)

```
from typing import List, Tuple, Any
from tools.db_connector import SQLiteConnector

connector = SQLiteConnector()


def _exec_sql(sql: str) -> Tuple[List[str], List[Tuple[Any, ...]]]:
    cur = connector.conn.cursor()
    cur.execute(sql)
    rows = cur.fetchall()
    cols = [d[0] for d in cur.description] if cur.description else []
    return cols, rows

def _format_table(cols: List[str], rows: List[Tuple[Any, ...]]) -> str:
    if not cols:
        return "No result columns."
    header = "|".join(cols)
    if not rows:
        return header  # header only when no matches
    lines = [header]
    for r in rows:
        lines.append("|".join("" if v is None else str(v) for v in r))
    return "\n".join(lines)

def read_database_tool(sql: str) -> str:

    if not isinstance(sql, str) or not sql.strip():
        return "Error executing SQL: empty query."

    try:
        cols, rows = _exec_sql(sql)
        return _format_table(cols, rows)
```

```
    except Exception as e:
        return f"Error executing SQL: {e}"
```

📦 Supplier Search Tool (`tools/web_searcher.py`)

```
import os
from langchain_tavily import TavilySearch
from config import TAVILY_API_KEY

# Initialize Tavily search tool
search_tool = TavilySearch(
    api_key=TAVILY_API_KEY,
    max_results=3,
    include_answer=True
)

def web_search_tool(query: str) -> str:

    if not query:
        return "No query provided."

    try:
        results = search_tool.invoke({"query": query})

        if not results or not results.get("results"):
            return f"No results found for '{query}'."

        output = f"Search results for **{query}**:\n"
        for result in results["results"]:
            title = result.get("title", "No title")
            url = result.get("url", "")
            snippet = result.get("content", "")
            output += f"- \\[{title}\\]({url})\n  {snippet}\n\n"

        return output.strip()

    except Exception as e:
        return f"Error during search: {str(e)}"
```

📧 Email Tool (`tools/email_sender.py`)

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.application import MIMEApplication
from typing import Optional
from config import (
    AGENT_EMAIL_ADDRESS,
    AGENT_EMAIL_PASSWORD,
)

# If not in config, define fallbacks:
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
```

```
def send_email(
    to_address: str,
    subject: str,
    body: str,
    attachment_path: Optional[str] = None
) -> str:
    """Send a plaintext email with optional attachment."""
    if not AGENT_EMAIL_ADDRESS or not AGENT_EMAIL_PASSWORD:
        raise ValueError("Missing email credentials (AGENT_EMAIL_ADDRESS or AGENT_EMAIL_PASSWORD).")

    msg = MIMEMultipart("alternative")   # ✅ use "alternative" to support HTML + plain text fallback
    msg["From"] = AGENT_EMAIL_ADDRESS
    msg["To"] = to_address
    msg["Subject"] = subject
    msg.attach(MIMEText(body, "plain"))
    msg.attach(MIMEText(body, "html"))

    if attachment_path:
        with open(attachment_path, "rb") as f:
            part = MIMEApplication(f.read(), Name=attachment_path)
        part["Content-Disposition"] = f'attachment; filename="{attachment_path}"'
        msg.attach(part)
    with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:
        server.starttls()
        server.login(AGENT_EMAIL_ADDRESS, AGENT_EMAIL_PASSWORD)
        server.send_message(msg)

    return "Email sent successfully."


def send_email_tool(input_str: str) -> str:

    from config import BOSS_EMAIL_ADDRESS
    subject, body = [x.strip() for x in input_str.split("||", 1)]

    try:
        result = send_email(
            to_address=BOSS_EMAIL_ADDRESS,
            subject=subject,
            body=body
        )
        return result
    except Exception as e:
        return f"Failed to send email: {e}"
```

🧠 Log Tracker (`tools/log_tracker.py`)

```
import sqlite3
from datetime import datetime
from config import DB_PATH


def track_log(action: str, details: str = "") -> str:

    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
```

```
    timestamp = datetime.now().isoformat()
    cursor.execute(
        "INSERT INTO logs (timestamp, action, details) VALUES (?, ?, ?)",
        (timestamp, action, details)
    )

    conn.commit()
    conn.close()

    return f"Logged action: {action}"


def track_log_tool(input: str) -> str:
    # Expect input like "Queried inventory | Checked products below threshold"
    try:
        action, details = input.split(" | ", 1)
    except ValueError:
        action, details = input, ""
    return track_log(action.strip(), details.strip())
```

Main `main.py`:

Add the following to main.py and run it.

```
from agents.inventory_agent import run_inventory_agent

run_inventory_agent(
    '''
    check our inventory, identify low stock items where sum(quantity) below threshold level,
    search for suppliers for our low stock items if there is any lowstock item.
    send me an email summary of the low stock items and the suppliers with links.
    '''
)
```

Create another entry point to the program **chat.py**.

```
from agents.inventory_agent import run_inventory_agent


print("What can I help you today. (Type 'exit' to exit.)")
while True:
    user_input = input("You: ").strip()
    if user_input.lower() in {"exit", "quit"}:
        print("Inventory Agent: Bye!")
        break
    answer = run_inventory_agent(user_input)
    print(f"Inventory Agent: {answer}")
```

Create a batch file (run_inventory_agent.bat) prefereably outside the project folder and add the following and run it by double clicking the batch file.

```
@echo off
REM Run Smart Inventory Agent
cd C:\...path to your project folder\smart-inventory-agent-final
call venv\Scripts\activate.bat
python main.py
```

Use windows Task Scheduler to Schedule the batch file run everyday.