Code Readme File

Real Estate Simulation Engine

1. Project Overview

This project is a smart tool built to tackle a real estate challenge. You give it a property scenario (like a house that's not selling), and it uses a team of AI agents to figure out what's wrong and what to do next. In the end, it gives you a simple data file (JSON) and a nicely formatted PDF report with its findings.

The engine does more than just the basics. It searches the web for live market data, uses a smart scoring system to evaluate strategies, and is built with a team of AI agents that work together using LangChain.

2. What It Can Do

- **A Team of AI Specialists:** Instead of one AI trying to do everything, this tool uses a chain of nine different AI agents. Each one has a specific job, like being an analyst, a strategist, or a report writer. This "assembly line" approach leads to smarter, more reliable advice.
- **Live Web Search (Powered by Tavily):** The AI doesn't just use its stored knowledge. It actively searches the web for the latest market data using the **Tavily Search API**. This is a critical component that dramatically improves the simulation's performance, allowing it to generate strategies based on the specific dynamics of the property's location instead of just offering generic advice.
- **Smart Strategy Scoring:** It doesn't just list ideas. It carefully scores each potential strategy based on its likely impact, speed, and cost/risk, so you can see *why* certain actions are recommended.
- **Multiple Output Formats:** You get the results in two ways: a clean JSON file for any other software to use, and a professional PDF memo that's easy to read and share.
- **Easy to Modify:** The code is organized neatly, so it's straightforward to change or add new features later on.

3. How the Project is Organized

Here's how the project is laid out:

- `main.py`: This is the main script that kicks everything off. It manages the team of analytical AI agents.
- `report_generator.py`: This file handles the final step—taking all the analysis and having an AI write the professional PDF report.
- `helper_classes.py`: This defines the data structures (using Pydantic) that the agents use to pass information to each other reliably.
- `input.json`: This is where you set up the property scenario you want to analyze.
- `requirements.txt`: A list of all the Python libraries you need to install.
- `.env`: A file you'll create to keep your API keys safe.

4. How the AI "Thinks"

Think of this tool as an assembly line for making smart decisions. Here's how it works, step-by-step:

1. **The Researcher:** First, an agent figures out the best things to search for on the web to understand the market.

2. **The Analyst:** It takes the search results and does a deep dive into the market and the specific property.
3. **The Diagnostician:** This one reads the analysis and pinpoints the single biggest problem.
4. **The Strategist:** Based on the problem, this agent brainstorms a list of creative, actionable solutions.
5. **The Evaluator:** This agent takes the list of strategies and does a detailed pro/con analysis on each one, giving them scores for impact, speed, and cost/risk.
6. **The Parser:** It cleans up the evaluator's text analysis and turns it into structured data that's easy for the other agents to use.
7. **The Finalizer:** This agent takes the structured data, picks the top 3 strategies, and creates the final `output.json` file.
8. **The Behavioural Coach:** This one gives advice on the "human side" of things—how the seller and agent should act to get the best result.
9. **The Report Writer:** Finally, this agent takes everything from the previous steps and writes the polished, easy-to-read PDF memo.

---

5. Setup and Installation

Follow these steps to get the project ready to run:

1. **Clone the Repository**:

   ```
   git clone <your-repo-url>
   cd <your-repo-name>
   ```

2. **Create a Virtual Environment** (Recommended):

   ```
   python -m venv venv
   source venv/bin/activate  # On Windows, use `venv\\Scripts\\activate`
   ```

3. **Install Dependencies**:
   Install all the required libraries from the `requirements.txt` file.

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

4. **Set Up Environment Variables**:
   Create a file named `.env` in the project's main folder and add your API keys.

   For Azure:

   ```
   # .env file
   AZURE_API_KEY="your_azure_openai_api_key"
   AZURE_ENDPOINT="your_azure_openai_endpoint"
   TAVILY_API_KEY="your_tavily_ai_api_key"
   ```

   Note on OpenAI API: If you prefer to use the standard OpenAI API instead of Azure, you can set your key under the name `OPENAI_API_KEY`. The code to use this is already included but commented out in both `main.py` (line 37) and `report_generator.py` (line 167).

```
# .env file (for OpenAI)
OPENAI_API_KEY="your_openai_api_key"
TAVILY_API_KEY="your_tavily_ai_api_key"
```

Note on Tavily API: The `TAVILY_API_KEY` is required for the web search feature. You can get a free key with 1,000 searches per month from their website: https://www.tavily.com/

6. How to Run

Getting the simulation running is simple:

1. **Edit the Input Scenario**:
   Open the
   `input.json` file and change the `scenario`, `goal`, and `constraint` to match what you want to test.

   ```
   {
    "scenario": "£5.25M Knightsbridge townhouse unsold for 9 months",
    "goal": "Secure offer within 60 days",
    "constraint": "Do not reduce below £4.2M"
   }
   ```

2. **Run the Simulation**:
   From your terminal, run the
   `main.py` script and tell it to use your `input.json` file.

   ```
   python main.py input.json
   ```

7. What You Get

After the script runs, you'll find these new files in your project folder:

- **`output.json`**: A data file with the core results (diagnosis, recommended actions, and score). Perfect for feeding into other applications.
- **`simulation_memo.pdf`**: A polished, easy-to-read report designed for people. It explains the findings, the recommended strategy, and the reasoning behind it.
- **`final_result.pkl`**: A developer's file. It saves *everything* from the simulation run, which is super helpful for debugging or seeing how the AI "thought" at each step.

8. Design Choices (Modularity)

The project was built with future growth in mind, tackling the optional "modularity" goals:

- **Building Prompts:** Instead of having giant, messy prompts, the instructions for the AI are broken into smaller, reusable pieces. For a bigger project, this would be built into a full system for managing and testing different prompts.
- **Formatting the Output:** The tool is careful about how it handles data. It uses special data classes (`Pydantic`) to make sure information is structured correctly between agents. Then, a final `ReportGenerator` agent makes the information look good for the final report.
- **Scoring Strategies:** The scoring system is its own separate logic. It uses a weighted formula to score each strategy, and this logic could easily be tweaked in the future if priorities change.

- **Tracking Versions:** To make sure results can be reproduced and debugged, the tool is set up to be run with version control (like Git). It also saves a `final_result.pkl` file, which acts as a "black box" recorder for everything that happened during a specific run.