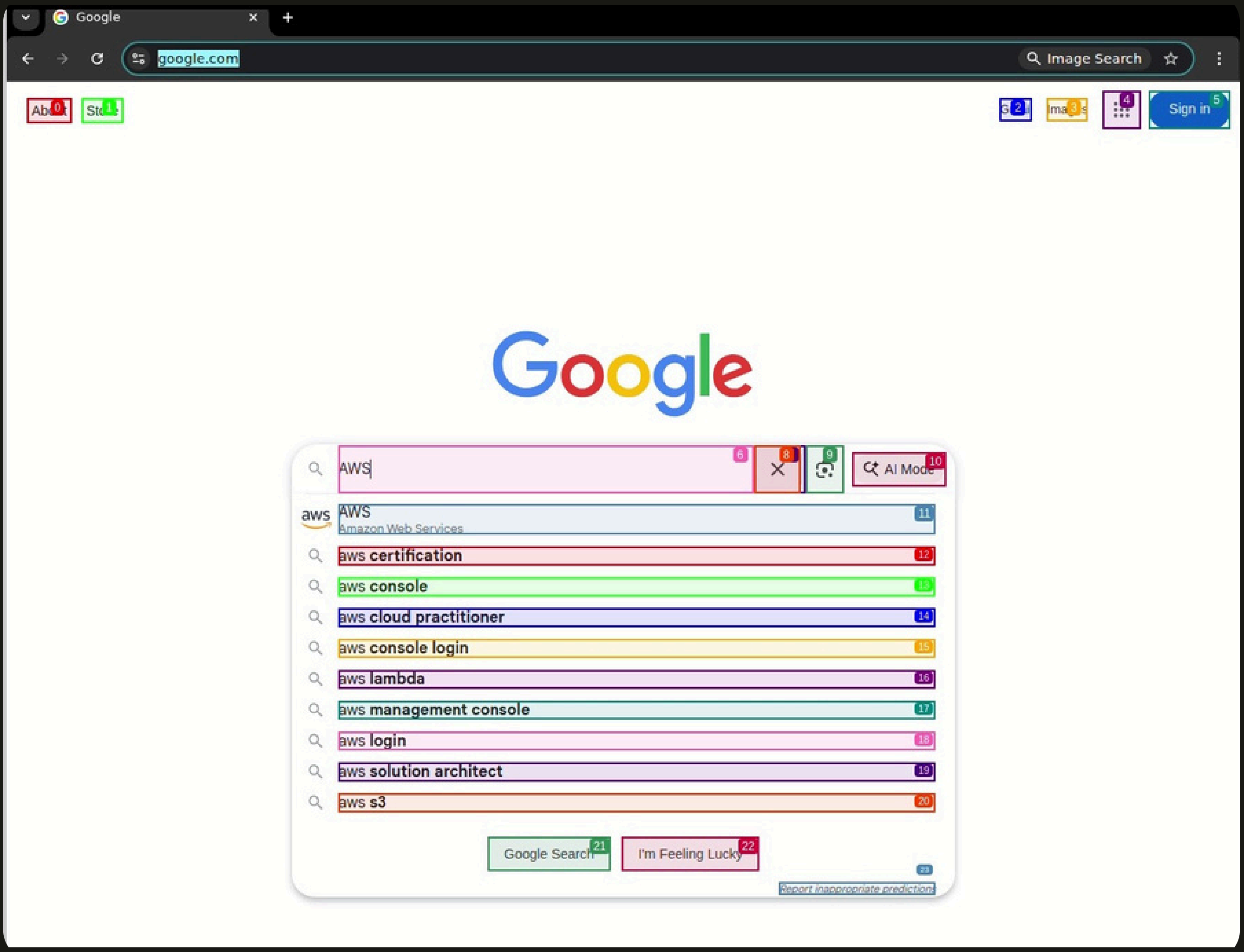
~~Linkedin~~

Automate Your Browser with AI Agents

DI Dineshraj Dhanapathy



Introducing Today's Project!

In this project, I will demonstrate how to set up a local environment by installing Python, Pip, UV, and Git, followed by configuring Browser Use's WebUI for browser automation. I'm doing this project to learn how to run automated browser tasks, write effective prompts, and explore how agents can perform real-world tasks like logging into LinkedIn. This hands-on experience will help me understand agent workflows and improve my automation and prompt engineering skills.

Tools and concepts

Services I used were WebUI, Playwright for browser automation, and the Gemini AI model for decision-making. Key concepts I learnt include setting up Python virtual environments, using API keys for secure authentication, configuring browsers with personal profiles, writing effective prompts for agents, troubleshooting errors, and understanding how browser engines like Chromium work. These skills helped me build, run, and refine AI browser agents to complete complex web tasks.

Project reflection

This project took me approximately a few hours to complete, as I spent time setting up WebUI, configuring the environment, and testing different tasks with the agent. The most challenging part was troubleshooting errors when prompts weren’t clear or when websites had complex structures that blocked automation. It was most rewarding to see the agent successfully complete tasks like navigating, extracting data, and using my own browser, proving the setup worked end-to-end.

I did this project today to strengthen my hands-on skills with WebUI, browser automation, and AI-driven agents, as I wanted practical experience beyond theory. My goal was to learn how to set up the environment, connect an AI model, and automate real web tasks while troubleshooting errors. This project met my goals because I not only practiced technical setup but also learned how to refine prompts, use personal

Development Environment

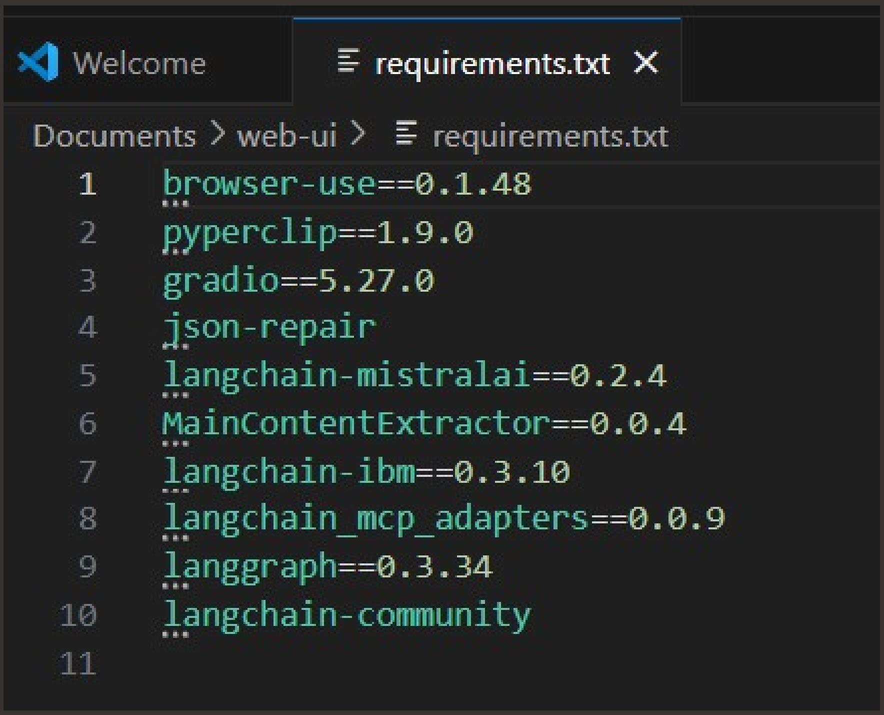
WebUI is an open-source interface for Browser Use that enables you to build AI-powered browser agents capable of navigating websites, clicking buttons, filling out forms, extracting data, and making decisions on next steps like a smart assistant. To retrieve WebUI's code, you need to download its source from the official repository, set up a Python virtual environment for managing dependencies, and then run WebUI to start using its features.

I installed Python because it is essential for running scripts and automation tasks in Linux. I installed pip because it is the package manager for Python, allowing me to easily install and manage libraries. I installed uv because it helps manage Python virtual environments efficiently, keeping dependencies isolated. I installed Git because it is a powerful version control system that enables me to track changes, collaborate on projects, and manage code repositories effectively.

I set up a virtual environment by creating an isolated Python workspace where only the required dependencies for WebUI were installed. A virtual environment is helpful for keeping project libraries separate from system-wide packages, avoiding version conflicts, and ensuring consistency across setups. It also makes it easier to manage, update, or remove dependencies without affecting other projects, providing a clean and reliable environment for running and testing the agent.

DI Dineshraj Dhanapathy

Linkedin



```
browser-use==0.1.48
pyperclip==1.9.0
gradio==5.27.0
json-repair
langchain-mistralai==0.2.4
MainContentExtractor==0.0.4
langchain-ibm==0.3.10
langchain_mcp_adapters==0.0.9
langgraph==0.3.34
langchain-community

```

Configuring My API

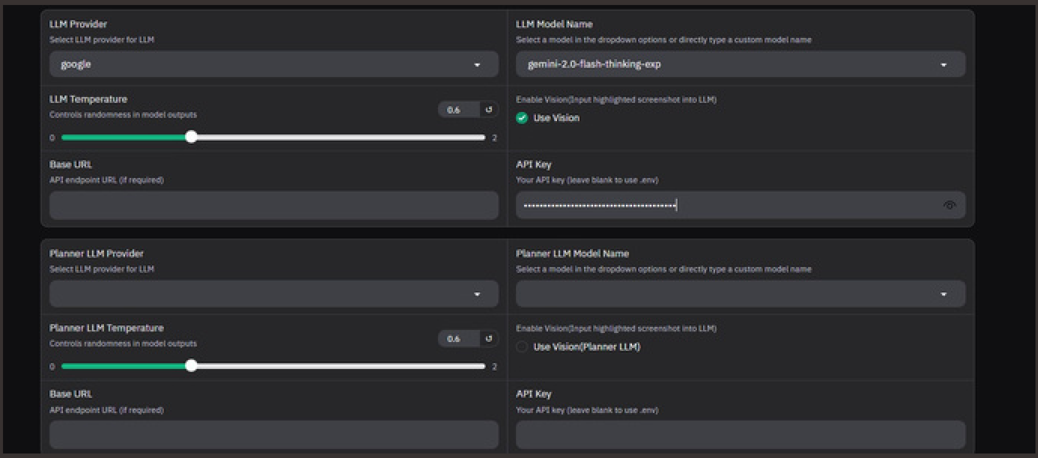
An API key is needed because it acts as a secure access pass that authorizes WebUI to connect with Google AI Studioś' models. It helps WebUI by enabling communication with the AI model, which serves as the brain of the browser agent. With this connection, WebUI can plan and execute tasks such as navigating websites, clicking buttons, filling forms, extracting data, and making decisions on the next steps to successfully complete the given task.

I set up my API key by generating it in Google AI Studio and ensuring it was copied and stored securely to prevent unauthorized access. Then I configured WebUI to use it by adding the key to the applicationś' environment variables or configuration file, allowing WebUI to authenticate with Googleś' AI services. This setup ensures WebUI can connect to the AI model, process tasks, and execute intelligent browser actions reliably and securely.

DI

Dineshraj Dhanapathy

Linkedin



LLM Provider
Select LLM provider for LLM
google

LLM Model Name
Select a model in the dropdown options or directly type a custom model name
gemini-2.0-flash-thinking-exp

LLM Temperature
Controls randomness in model outputs
0.6
0 2

Enable Vision(Input highlighted screenshot into LLM)
✓ Use Vision

Base URL
API endpoint URL (if required)

API Key
Your API key (leave blank to use .env)
•••••••••••••••••••••••••••••••••••••••

Planner LLM Provider
Select LLM provider for LLM

Planner LLM Model Name
Select a model in the dropdown options or directly type a custom model name

Planner LLM Temperature
Controls randomness in model outputs
0.6
0 2

Enable Vision(Input highlighted screenshot into LLM)
○ Use Vision(Planner LLM)

Base URL
API endpoint URL (if required)

API Key
Your API key (leave blank to use .env)

Running The Agent

My first agent's task was to search for "AWS" on Google and find the first URL result, after which the new browser window collapsed automatically. The agent by default launches Chromium, which is the open-source browser engine powering Google Chrome, Edge, and Brave. Chromium acts as both a browser and an engine, capable of reading webpage code, rendering it visually, and handling interactions like clicks and navigation to complete the task effectively.

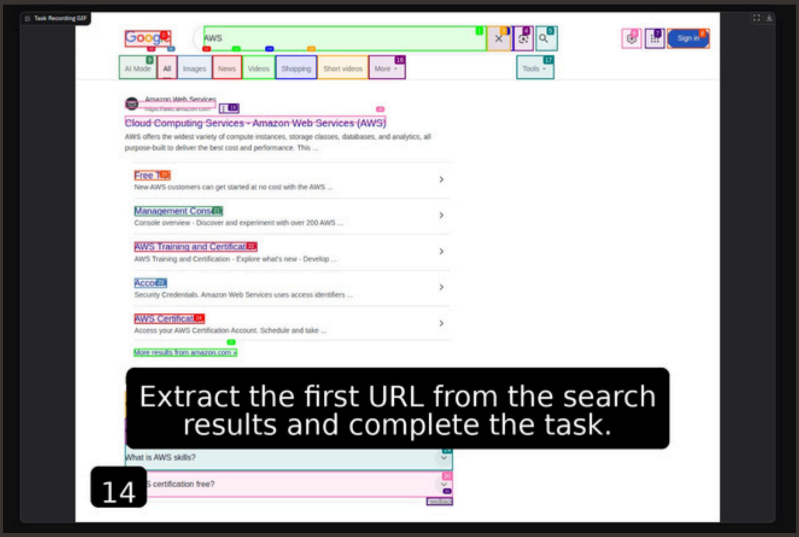
The agent interacts with the browser by using Playwright to highlight and control elements like buttons, links, and forms, enabling automated clicks, typing, and navigation. I can see this when colored boxes appear around interactive elements, showing what the agent is analyzing in real time. Behind the scenes, the Gemini AI model decides which actions to take, allowing the agent to complete tasks efficiently and move step by step toward the final goal.

The Results tab shows the outcome of the agentś' actions, including the data it extracted, the steps it took, and the final result of the assigned task. This is helpful because it gives a clear summary of what the agent accomplished, making it easier to verify accuracy, track progress, and debug any issues. By reviewing the Results tab, I can confirm whether the agent successfully met the task requirements or if adjustments are needed for better performance.

DI

Dineshraj Dhanapathy

Linkedin


Task Recording GIF

AWS

AI Mode | All | Images | News | Videos | Shopping | Short videos | More | Tools

Amazon Web Services
https://aws.amazon.com

Cloud Computing Services - Amazon Web Services (AWS)

AWS offers the widest variety of compute instances, storage classes, databases, and analytics, all purpose-built to deliver the best cost and performance. This ...

Free T
New AWS customers can get started at no cost with the AWS ...

Management Cons
Console overview · Discover and experiment with over 200 AWS ...

AWS Training and Certificat
AWS Training and Certification · Explore what's new · Develop ...

Acco
Security Credentials. Amazon Web Services uses access identifiers ...

AWS Certificat
Access your AWS Certification Account. Schedule and take ...

More results from amazon.com »

Extract the first URL from the search results and complete the task.

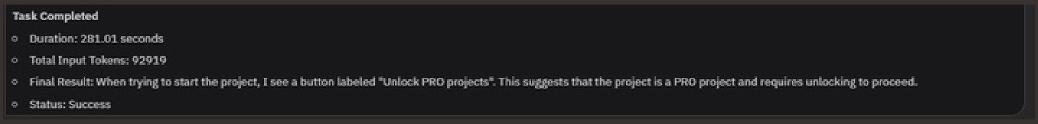
What is AWS skills?

S certification free?

14

Advanced Navigation

I also tried telling the agent to start the project, and it reported that the button labeled "Unlock PRO projects" appeared, indicating the project is part of the PRO tier and requires unlocking to continue. This means access is restricted without the proper subscription or permission. The agent identified that this lock prevents direct execution of the project and highlighted the need to upgrade or enable PRO features before proceeding with running or testing the project fully.



```
Task Completed
○ Duration: 281.01 seconds
○ Total Input Tokens: 92919
○ Final Result: When trying to start the project, I see a button labeled "Unlock PRO projects". This suggests that the project is a PRO project and requires unlocking to proceed.
○ Status: Success
```

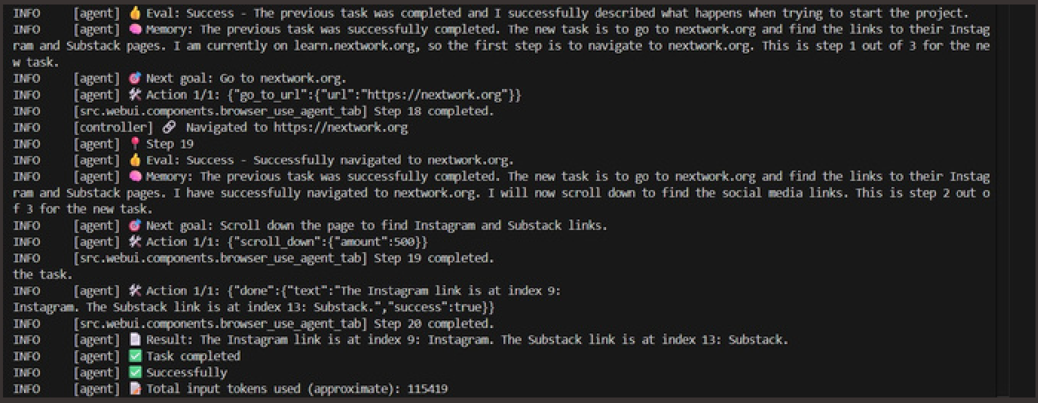
Debugging and Logs

When my agent encounters an error, three common reasons are unclear task descriptions without a defined output format, complex or dynamic website structures that make elements hard to locate, and websites blocking automated access. To fix this, I can refine prompts to specify the expected result, guide the agent with more context, or adjust actions like scrolling and clicking. In some cases, adding retries, using alternative selectors, or handling authentication may also resolve the issue.

The terminal logs show a detailed step-by-step record of the agentš' actions, including its goals, navigation, scrolling, evaluations, memory updates, and final results. They display which URLs were visited, what steps were completed, and the success of each task, such as locating Instagram and Substack links. This helps me understand how the agent processes instructions, verifies progress, and reaches completion, while also providing transparency for troubleshooting and improving reliability.

DI Dineshraj Dhanapathy

Linkedin



```
INFO     [agent] 👍 Eval: Success - The previous task was completed and I successfully described what happens when trying to start the project.
INFO     [agent] 🧠 Memory: The previous task was successfully completed. The new task is to go to nextwork.org and find the links to their Instag
ram and Substack pages. I am currently on learn.nextwork.org, so the first step is to navigate to nextwork.org. This is step 1 out of 3 for the ne
w task.
INFO     [agent] 🎯 Next goal: Go to nextwork.org.
INFO     [agent] 🛠 Action 1/1: {"go_to_url":{"url":"https://nextwork.org"}}
INFO     [src.webui.components.browser_use_agent_tab] Step 18 completed.
INFO     [controller] 🔗  Navigated to https://nextwork.org
INFO     [agent] 📍 Step 19
INFO     [agent] 👍 Eval: Success - Successfully navigated to nextwork.org.
INFO     [agent] 🧠 Memory: The previous task was successfully completed. The new task is to go to nextwork.org and find the links to their Instag
ram and Substack pages. I have successfully navigated to nextwork.org. I will now scroll down to find the social media links. This is step 2 out o
f 3 for the new task.
INFO     [agent] 🎯 Next goal: Scroll down the page to find Instagram and Substack links.
INFO     [agent] 🛠 Action 1/1: {"scroll_down":{"amount":500}}
INFO     [src.webui.components.browser_use_agent_tab] Step 19 completed.
the task.
INFO     [agent] 🛠 Action 1/1: {"done":{"text":"The Instagram link is at index 9:
Instagram. The Substack link is at index 13: Substack.","success":true}}
INFO     [src.webui.components.browser_use_agent_tab] Step 20 completed.
INFO     [agent] 📄 Result: The Instagram link is at index 9: Instagram. The Substack link is at index 13: Substack.
INFO     [agent] ✅ Task completed
INFO     [agent] ✅ Successfully
INFO     [agent] 📝 Total input tokens used (approximate): 115419
```

Advanced Browser Configuration

In this project extension, I will configure the agent to use websites with my personal login, showing advanced agent setup skills. Using your own browser is useful because it allows the agent to access personalized data, saved sessions, and credentials that wouldn't be available in a default environment. This makes it possible to automate tasks on accounts I regularly use, ensuring the agent can interact with real, authenticated workflows while maintaining flexibility and realism in testing.

To use your own browser in WebUI, you have to update the Chrome configuration settings by specifying the correct paths. The CHROME_PATH setting tells WebUI where the Google Chrome executable is located so it can launch the browser, while CHROME_USER_DATA setting tells it where to find your personal Chrome profile data, including logins, cookies, and preferences. Together, these settings let WebUI run tasks using your own browser environment with saved sessions and credentials.

I chose to test my agent with logging into a personal account-enabled website using my own browser settings. The agent was able to access the site with my saved session, navigate through the pages, and perform actions like clicking menus and extracting information. This showed how using my browser allows the agent to leverage existing logins and preferences, making it possible to automate real workflows securely and efficiently without repeatedly entering credentials.