**Full Code with Line-by-Line Explanation**

import time

* **Purpose**: Imports the time module to use time.sleep() for pauses (e.g., waiting for cart updates or shipping checks).
* **Context**: Used in add\_to\_cart and track\_shipping to handle dynamic page loads.

from typing import TypedDict

* **Purpose**: Imports TypedDict to define a typed dictionary for the agent’s state.
* **Context**: Ensures ShoppingState has specific keys with types, helping LangGraph manage state transitions.

from langgraph.graph import StateGraph, END

* **Purpose**:
  + StateGraph: Class to build the agentic workflow as a graph of nodes.
  + END: Marker for the graph’s termination point.
* **Context**: Structures your agent’s flow: search → cart → payment → track → done.

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

* **Purpose**: Imports Selenium components:
  + webdriver: Controls the Chrome browser.
  + By: Locates elements (e.g., by ID or class).
  + Service: Configures ChromeDriver’s path.
  + WebDriverWait: Waits for page elements to load.
  + EC (expected\_conditions): Conditions like "element is clickable."
* **Context**: Drives browser actions (e.g., clicking "Add to Cart").

from duckduckgo\_search import DDGS

* **Purpose**: Imports DuckDuckGo’s search client (DDGS) to find Gillette razor URLs.
* **Context**: Used in search\_razor to query Amazon product pages.

from langchain\_groq import ChatGroq

* **Purpose**: Imports Grok’s integration via langchain\_groq for LLM capabilities.
* **Context**: Adds Grok (Mixtral 8x7B) to pick the best razor and offer payment tips.

llm = ChatGroq(

groq\_api\_key="your\_grok\_api\_key\_here",

model\_name="mixtral-8x7b-32768",

temperature=0.2

)

* **Purpose**: Initializes the Grok LLM client:
  + groq\_api\_key: Your API key from [console.groq.com](https://console.groq.com).
  + model\_name: Uses Mixtral 8x7B (Grok proxy), fast and reasoning-capable.
  + temperature=0.2: Low randomness for precise outputs.
* **Context**: Global LLM instance for search\_razor and wait\_for\_payment.

class ShoppingState(TypedDict):

driver: webdriver.Chrome

product\_url: str

in\_cart: bool

payment\_done: bool

shipping\_status: str

tracking\_url: str

search\_results: list

* **Purpose**: Defines the agent’s state as a typed dictionary:
  + driver: Chrome browser instance.
  + product\_url: URL of the selected razor.
  + in\_cart: Tracks if the item’s added.
  + payment\_done: Flags user checkout completion.
  + shipping\_status: Order state ("pending," "shipped," "delivered").
  + tracking\_url: Shipping link.
  + search\_results: Stores DuckDuckGo results for potential reuse.
* **Context**: Passed between nodes, updated at each step.

def setup\_driver():

service = Service(executable\_path="C:/agenticAI/AgenticAIWorkspace/chromedriver.exe")

driver = webdriver.Chrome(service=service)

return driver

* **Purpose**: Sets up Selenium’s ChromeDriver:
  + service: Points to your ChromeDriver path.
  + driver: Launches Chrome with that driver.
  + return: Returns the browser instance.
* **Context**: Called once to initialize the browser for all nodes.

def search\_razor(state: ShoppingState) -> ShoppingState:

print("Searching for Gillette razor...")

with DDGS() as ddgs:

query = "razor site:amazon.com Gillette inurl:/dp/"

results = ddgs.text(query, max\_results=5)

* **Purpose**: Searches for a Gillette razor:
  + print: Logs the step.
  + with DDGS(): Opens a DuckDuckGo search context.
  + query: Targets Amazon product pages (/dp/).
  + results: Fetches up to 5 results.

if not results:

print("No results found.")

return {"product\_url": None, "in\_cart": False, "search\_results": []}

* **Purpose**: Handles no results:
  + if not results: Checks if the search failed.
  + print: Logs the failure.
  + return: Updates state with no URL.

options = "\n".join([f"{i+1}. {r['title']} - {r['href']}" for i, r in enumerate(results)])

prompt = f"From these Amazon Gillette razor options, pick the best product URL (must have /dp/):\n{options}\nReturn only the URL."

response = llm.invoke(prompt)

selected\_url = response.content.strip()

* **Purpose**: Uses Grok to pick the best razor:
  + options: Formats results (e.g., "1. Gillette Mach3 - https://...").
  + prompt: Asks Grok to select a URL.
  + response: Gets Grok’s answer.
  + selected\_url: Extracts the raw URL.

print(f"Grok selected: {selected\_url}")

if "amazon.com" in selected\_url and "/dp/" in selected\_url:

return {"product\_url": selected\_url, "in\_cart": False, "search\_results": results}

else:

print("Grok picked an invalid URL, falling back to first valid result.")

for result in results:

if "amazon.com" in result["href"] and "/dp/" in result["href"]:

return {"product\_url": result["href"], "in\_cart": False, "search\_results": results}

* **Purpose**: Validates Grok’s choice:
  + print: Logs the selection.
  + if: Ensures it’s an Amazon product URL.
  + else: Fallback to first valid result if Grok fails.
  + return: Updates state with URL and results.

return {"product\_url": None, "in\_cart": False, "search\_results": []}

* **Purpose**: Default return if no valid URL is found.

def add\_to\_cart(state: ShoppingState) -> ShoppingState:

if not state["product\_url"]:

print("No product URL available.")

return state

* **Purpose**: Checks for a URL:
  + if: Skips if no URL from search\_razor.
  + return: Returns unchanged state.

print("Adding Gillette razor to cart...")

driver = state["driver"]

driver.get(state["product\_url"])

* **Purpose**: Loads the product page:
  + driver: Gets the browser from state.
  + get: Navigates to the URL.

try:

add\_button = WebDriverWait(driver, 15).until(

EC.element\_to\_be\_clickable((By.ID, "add-to-cart-button"))

)

add\_button.click()

time.sleep(2)

print("Successfully added to cart.")

return {"in\_cart": True}

* **Purpose**: Adds the item to cart:
  + try: Handles potential errors.
  + WebDriverWait: Waits up to 15s for the button.
  + By.ID: Finds "Add to Cart" by its ID.
  + click: Adds the razor.
  + sleep: Waits 2s for cart update.
  + return: Marks in\_cart as True.

except Exception as e:

print(f"Error adding to cart: {e}")

print(f"Current URL: {driver.current\_url}")

return {"in\_cart": False}

* **Purpose**: Catches failures:
  + print: Logs error and URL for debugging.
  + return: Marks in\_cart as False.

def wait\_for\_payment(state: ShoppingState) -> ShoppingState:

if not state["in\_cart"]:

print("Cart is empty, stopping.")

return state

* **Purpose**: Ensures cart has an item:
  + if: Skips if add\_to\_cart failed.

driver = state["driver"]

driver.get("https://www.amazon.com/gp/cart/view.html")

print("Please log in, enter your credit card details, and complete checkout.")

* **Purpose**: Opens cart for manual checkout:
  + get: Navigates to cart page.
  + print: Instructs user.

prompt = "User is at Amazon checkout. Provide a concise tip for smooth payment."

tip = llm.invoke(prompt).content.strip()

print(f"Grok tip: {tip}")

* **Purpose**: Grok gives a payment tip:
  + prompt: Asks for advice.
  + tip: Gets and cleans Grok’s response.
  + print: Shows the tip (e.g., "Use a saved card").

input("Press Enter after payment to resume tracking...")

return {"payment\_done": True}

* **Purpose**: Waits for user:
  + input: Pauses until Enter is pressed.
  + return: Marks payment complete.

def track\_shipping(state: ShoppingState) -> ShoppingState:

if not state["payment\_done"]:

print("Payment not completed, stopping.")

return state

* **Purpose**: Ensures payment happened.

driver = state["driver"]

driver.get("https://www.amazon.com/gp/your-account/order-history")

print("Tracking order status...")

max\_attempts = 10

attempt = 0

* **Purpose**: Starts tracking:
  + get: Loads order history.
  + max\_attempts: Limits retries to 10 (~5min).

while attempt < max\_attempts:

try:

status = WebDriverWait(driver, 15).until(

EC.presence\_of\_element\_located((By.CLASS\_NAME, "order"))

)

status\_text = status.text.lower()

* **Purpose**: Checks order status:
  + while: Loops until shipped or timeout.
  + status: Waits for order element.
  + status\_text: Gets text (e.g., "shipped").

if "shipped" in status\_text:

tracking\_link = driver.find\_element(By.XPATH, "//a[contains(text(), 'Track')]")

print("Order has shipped!")

return {

"shipping\_status": "shipped",

"tracking\_url": tracking\_link.get\_attribute("href")

}

* **Purpose**: Handles "shipped":
  + if: Detects shipment.
  + tracking\_link: Finds tracking URL.
  + return: Updates state with tracking.

elif "delivered" in status\_text:

print("Order delivered!")

return {"shipping\_status": "delivered", "tracking\_url": ""}

* **Purpose**: Handles "delivered":
  + elif: Detects delivery.
  + return: Updates state.

else:

print("Order not yet shipped. Checking again in 30 seconds...")

time.sleep(30)

driver.refresh()

* **Purpose**: Waits if not shipped:
  + sleep: Pauses 30s.
  + refresh: Reloads page.

except Exception as e:

print(f"Error tracking: {e}")

time.sleep(30)

driver.refresh()

attempt += 1

* **Purpose**: Handles errors and retries:
  + except: Catches issues (e.g., page load fail).
  + attempt: Increments retry counter.

print("Tracking timed out.")

return {"shipping\_status": "pending"}

* **Purpose**: Ends if max attempts hit.

def build\_workflow():

workflow = StateGraph(ShoppingState)

workflow.add\_node("search", search\_razor)

workflow.add\_node("cart", add\_to\_cart)

workflow.add\_node("payment", wait\_for\_payment)

workflow.add\_node("track", track\_shipping)

workflow.set\_entry\_point("search")

workflow.add\_edge("search", "cart")

workflow.add\_edge("cart", "payment")

workflow.add\_edge("payment", "track")

workflow.add\_edge("track", END)

return workflow.compile()

* **Purpose**: Builds the LangGraph workflow:
  + StateGraph: Initializes with ShoppingState.
  + add\_node: Registers each function.
  + set\_entry\_point: Starts at search.
  + add\_edge: Defines flow: search → cart → payment → track → END.
  + compile: Creates executable graph.

def run\_shopping\_agent():

driver = setup\_driver()

initial\_state = {

"driver": driver,

"product\_url": None,

"in\_cart": False,

"payment\_done": False,

"shipping\_status": "pending",

"tracking\_url": "",

"search\_results": []

}

* **Purpose**: Sets up and runs the agent:
  + driver: Initializes browser.
  + initial\_state: Defines starting state.

try:

app = build\_workflow()

final\_state = app.invoke(initial\_state)

print("\nFinal State:")

print(f"Product URL: {final\_state['product\_url']}")

print(f"In Cart: {final\_state['in\_cart']}")

print(f"Payment Done: {final\_state['payment\_done']}")

print(f"Shipping Status: {final\_state['shipping\_status']}")

print(f"Tracking URL: {final\_state['tracking\_url']}")

finally:

driver.quit()

print("Agent completed.")

* **Purpose**: Executes and cleans up:
  + try: Runs the workflow.
  + invoke: Processes state through nodes.
  + print: Shows final state.
  + finally: Closes browser.

if \_\_name\_\_ == "\_\_main\_\_":

run\_shopping\_agent()

* **Purpose**: Runs the script if executed directly.

**How It Flows**

1. **Search**: Grok picks a Gillette razor URL from DuckDuckGo.
2. **Cart**: Selenium adds it to your Amazon cart.
3. **Payment**: Grok advises, you manually pay.
4. **Track**: Selenium monitors until shipped.