Fetch.ai SDK Guide: Search Agent

This is currently a work in progress and may be subject to further updates and revisions. This guide explains the implementation of a Search Agent and Query Agent using the Fetch.ai SDK. These Agents work together to handle user queries, search the Fetch.ai network for relevant Agents, and return results.

Below, we outline theSearch Function and theComplete Agent Implementation in details.

Installation

Once you successfully have Python installed, you can start setting up your environment.

Create a new folder; call itfetchai-search-agent :

mkdir fetchai-search-agent && cd fetchai-search-agent Then, install the required libraries:

poetry install uagents fetchai openai

Search Function

The Search Function is the core of the Search Agent. It interacts with the Fetch.ai network to find Agents matching a specific query and sends them a message.

Here's how it works:

Functionalities

- Search the Fetch.ai Network
- : Thefetch.ai(query)
- method searches for Agents that match the user's query.
- Set Sender Identity
- : Each message is sent using a unique sender identity generated byIdentity.from_seed
- .
- Dynamic Payload
- : Uses OpenAI's GPT to generate payload based on the query.
- Send Messages
- : For every matching Agent, a payload is constructed and sent using thesend_message_to_agent
- function.

from fetchai import fetch from fetchai . crypto import Identity from fetchai . communication import send_message_to_agent from uuid import uuid4

def

search (query):

Search for agents matching the query

available_ais

fetch . ai (query)

Create sender identity for communication

sender_identity

Identity . from_seed ("search_sender_identity" , 0)

for ai in available_ais . get ('ais'):

Iterate through discovered agents

prompt

f """ you will take the following information: query= { query } . You must return a results according to the query"""

completion

client . chat . completions . create (model = "gpt-4o" , messages = [{ "role" : "user" , "content" : prompt}])

payload

{ "Response" : completion . choices [0]. message . content }

other_addr

ai . get ("address" , "")

Get agent's address

print (f "Sending a message to an AI agent at address: { other_addr } ")

Send the payload to the discovered agent

send_message_to_agent (sender = sender_identity, target = other_addr, payload = payload, session = uuid4 (),)

return

{ "status" :

"Agent searched" }

Complete Agent Implementation

The implementation involves two Agents:Query Agent andSearch Agent , which interact with each other as follows:

Query Agent

- It sends the user's query to the Search Agent on startup.

- it then waits for a response from the Search Agent.
- query_agent
- =
- Agent
- (name
- =
- "query_agent"
- , seed
- =
- "query_agent recovery phrase"
-)

Search Agent

-Receives the query, processes it using thesearch() function, and sends a response back to the Query Agent.

search_agent

Agent (name = "search_agent" , seed = "search_agent recovery phrase")

Overall script

fetchai-search-agent.py from fetchai import fetch from fetchai . crypto import Identity from fetchai . communication import (send_message_to_agent)

from openai import OpenAI

client

OpenAI ()

from uuid import uuid4 from uagents import Agent , Bureau , Context , Model

class

Query (Model): message :

str

class

Response (Model): status :

str

def

search (query): available_ais = fetch . ai (query) sender_identity = Identity . from_seed ("whatever i want this to be, but i am searching" , 0) print (f "[results] available ais : { available_ais } ") for ai in available_ais . get ('ais'):

prompt

f """ you will take the following information: query= { query } . You must return a results according to the query"""

completion

client . chat . completions . create (model = "gpt-4o" , messages = [{ "role" : "user" , "content" : prompt}])

payload

{ "Response" : completion . choices [0]. message . content }

other_addr

ai . get ("address" , "")

print (f "[INFO] Sending message to AI at address: { other_addr } ")

send_message_to_agent (sender = sender_identity, target = ai. get ("address" , ""), payload = payload, session = uuid4 ())

return

{ "status" :

"Agent searched" }

query_agent

Agent (name = "query_agent" , seed = "query_agent recovery phrase") search_agent =

Agent (name = "search_agent" , seed = "search_agent recovery phrase")

user_query

input ("Enter your query: ")

@query_agent . on_event ("startup") async

def

send_message (ctx : Context): ctx . logger . info ("[STARTUP] Query agent starting up and sending user query to search agent.") await ctx . send (search_agent.address, Query (message = user_query))

@search_agent . on_message (model = Query) async

def

sigmar_message_handler (ctx : Context ,

sender :

str ,

msg : Query): ctx . logger . info (f "[RECEIVED] Query received from { sender } . Message: ' { msg.message } '") results =

search (msg.message) ctx . logger . info ("[PROCESSING] Searching completed. Sending response back to the query agent.") await ctx . send (query_agent.address, Response (status = results["status"]))

@query_agent . on_message (model = Response) async

def

slaanesh_message_handler (ctx : Context ,

sender :

str ,

msg : Response): ctx . logger . info (f "[RECEIVED] Response received from search agent { sender } . Status: ' { msg.status } '")

bureau

Bureau () bureau . add (query_agent) bureau . add (search_agent)

if

name

==

"**main**" : print ("[INFO] Starting Bureau with Query Agent and Search Agent...") bureau . run ()

Running the Agent

We runfetchai-search-agent.py with the following commands:

python fetchai-search-agent.py

Expected output

The expected output from thefetchai-search-agent should be similar to the following:

Enter your query: Buy me a pair of shoes [INFO] Starting Bureau with Query Agent and Search Agent... INFO: [query_agent]: [STARTUP] Query agent starting up and sending user query to search agent. INFO: [search_agent]: [RECEIVED] Query received from agent1qdpstehd8x39n3jr0mas3adcy9d7rh4ss8wtw6euch0mq04tqu66kpfcu3q. Message: 'Buy me a pair of shoes' INFO:httpx:HTTP Request: POST https://agentverse.ai/v1/search/agents "HTTP/1.1 200 OK" [results] available ais : { "ais": [{ "address": "agent1qw7802t7qf98kg775k7f5v3f9h864c72eja2r94pumxnvyx3492xyzu8fmg", "name": "My AI's Name", "readme": "\nUsed for getting the nike shoes\n\n Used to get the nike shoes\n\n\nnike ai shoess\n\n \n question\n to gwt the nike shoese\n \n\n\n", "protocols": [{ "name": "", "version": "", "digest": "proto:a03398ea81d7aaaf67e72940937676eae0d019f8e1d8b5efbadfef9fd2e98bb2" }],... },...] } INFO:httpx:HTTP Request: POST https://api.openai.com/v1/chat/completions "HTTP/1.1 200 OK" [INFO] Sending message to AI at address: agent1qv5jr3ylluy45hzwftgctt0hnaa75h2m0ehflxdw8rz3720pjmaegy2447r {"version":1,"sender":"agent1qdtxzn2e0dg8y2v5y53p7frplt4w6wq36rfapv38g8x9ukgpc28fgfqnjug","target":"agent1qv5jr3ylluy45hzwftgctt0hnaa75h2m0ehflxdw8rz3720pjmaegy2447r","session":"924e26cb-bae2-4442-95a6-02292125c4f4","schema_digest":"model:708d789bb90924328daa69a47f7a8f3483980f16a1142c24b12972a2e4174bc6","protocol_digest":"proto:a03398ea81d7aaaf67e72940937676eae0d019f8e1d8b5e INFO:fetchai:Got response looking up agent endpoint https://staging-api.flockx.io/v1/chats/webhook_agent/ INFO:fetchai:Sent message to agent INFO: [search_agent]: [PROCESSING] Searching completed. Sending response back to the query agent. INFO: [bureau]: Starting server on http://0.0.0.0:8000 (Press CTRL+C to quit) INFO: [query_agent]: [RECEIVED] Response received from search agent agent1qgj8y2mswcc4jm275tsnq948fa7aqe8d9v0jd78h0nx9ak6v3fnxj6m6pkj. Status: 'Agent searched' Last updated on January 20, 2025

Was this page helpful?

You can also leave detailed feedback[on Github]

Chat protocol Introducing dialogues

On This Page

- Installation
- Search Function
- Functionalities
- Complete Agent Implementation
- Query Agent
- Search Agent
- Overall script
- Running the Agent
- Expected output
- Edit this page on github(opens in a new tab)