Quickstart with Fetch.ai SDK

The Fetch.ai SDK gives you easy access to the core components of the uAgents library.

Allowing you to create Agents outside the asynchronous library of uAgents. This is really great if you want to build synchronous programs but still have agentic functionalities.

Let's build two simpleAgent system using Flask, and OpenAI.

Installation

With Python installed, let's set up our environment.

Create a new folder; let's call itfetchai-tarot :

mkdir fetchai-tarot && cd fetchai-tarot Then, let's install our required libraries:

pip install flask openai fetchai We'll also need to set a couple of environment variables. In terminal export the following:

export OPENAI_API_KEY = "your_api_key_from_openai.com" export AGENTVERSE_KEY = "your_api_key_from_agentverse.ai" We have a guide to get anAgentverse api key

With that, let's get our Agents running.

The Tarot agent

The Tarot Agent is registers and listens on a webhook within a Flask app. This Agent uses OpenAI to create a Tarot reading from the received messages.

First, create a file and copy the following code in:

import json from flask import Flask , request from fetchai . communication import (send_message_to_agent , parse_message_from_agent) from fetchai . crypto import Identity from fetchai . registration import register_with_agentverse import os import sys from openai import OpenAI

client

OpenAI ()

app

Flask (**name**)

AGENTVERSE_KEY

os . environ . get ('AGENTVERSE_KEY' , "") if AGENTVERSE_KEY ==

"" : sys . exit ("Environment variable AGENTVERSE_KEY not defined")

You wouldn't normally want to expose the registration logic like this,

but it works for a nice demo.

@app . route ('/register' , methods = ['GET']) def

register ():

ai_identity

Identity . from_seed ("whatever i want this to be" , 0)

name

"tarot-agent"

readme

""" My AI's description of capabilities and offeringsMy AI returns your Tarot readingThe requirements your AI has for requests Date of birth I need your date of birthgender I need your gender """

the address in which your agent will respond to incoming messages

ai_webhook

os . environ . get ('AGENT_WEBHOOK' , "http://127.0.0.1:5000/webhook")

register_with_agentverse (ai_identity, ai_webhook, AGENTVERSE_KEY, name, readme,)

return

{ "status" :

"Agent registered" }

@app . route ('/webhook' , methods = ['POST']) def

webhook (): data = request . json try : message =

parse_message_from_agent (json. dumps (data)) except

ValueError

as e : return

{ "status" :

f "error: { e } " }

sender

message . sender payload = message . payload gender = payload . get ("gender" , None) date_of_birth = payload . get ("date of birth" , None)

ai_identity

Identity . from_seed ("whatever i want this to be" , 0)

if sender == ai_identity . address : print ("you are replying to yourself, just ignore.") return

{ "status" :

"Agent message processed" }

if gender is

None

or date_of_birth is

None : payload =

{ "err" :

"You need to provide gender and date of birth in the message to get a tarot reading" , } else : prompt =

f """ You are a Tarot teller, you will take the following information: gender= { gender } , date_of_birth= { date_of_birth } . You must return a tarot reading that is simple for someone to understand."""

completion

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

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

send_message_to_agent (sender = ai_identity, target = sender, payload = payload, session = data["session"]) return

{ "status" :

"Agent message processed" }

if

name

==

"**main**" : app . run (host = '0.0.0.0' , port = 5000 , debug = True) There's a lot to unpack there, but as this is a quickstart, let's just cover the essentials:

An Agent is identifiable by theirseed ; we use this to load in an Agent often withIdentity.from_seed("some str as your seed", 0)

Registration allows other Agents to find you, you can see this inregister_with_agentverse . We specify areadme which is a xml string so that LLMs can read this content in a more structured manner. Very useful for search in more complex demos.

/webhook is where our Agent accepts an incoming message. They will receive this message when another Agents search for it and match. Within this Agent, we get the data withdata = request.json and then we get some of the message objects, our Agent expects the payload to havedate of birth andgender as keys.

Finally, we create a message to be sent back to the sender:

send_message_to_agent(sender=ai_identity, target=sender, payload=payload, session=data["session"]) Sending messages to Agents is really simple. Depending on the other Agent, your payload may not need any structure at all.session is optional, and only useful for dialogues between Agents that require tracing, or are there are many steps in the dialogue.

The Search agent

The Search Agent searches for the Tarot Agents and sends them a payload ofgender anddate of birth and expects their tarot to be returned.

import json , os , sys from flask import Flask , request from fetchai import fetch from fetchai . communication import (send_message_to_agent , parse_message_from_agent) from fetchai . crypto import Identity from fetchai . registration import register_with_agentverse

from uuid import uuid4

app

Flask (**name**)

AGENTVERSE_KEY

os . environ . get ('AGENTVERSE_KEY' , "") if AGENTVERSE_KEY ==

"" : sys . exit ("Environment variable AGENTVERSE_KEY not defined")

@app . route ('/register' , methods = ['GET']) def

register ():

ai_identity

Identity . from_seed ("whatever i want this to be, but i am searching" , 0)

name

"tarot-search"

This is how you optimize your AI's search engine performance

readme

""" My AI's description of capabilities and offeringsPersonal agent that searches for my creator """

The webhook that your AI receives messages on.

ai_webhook

os . environ . get ('TAROT_AGENT_WEBHOOK' , "http://127.0.0.1:5002/webhook")

register_with_agentverse (ai_identity, ai_webhook, AGENTVERSE_KEY, name, readme,)

return

{ "status" :

"Agent registered" }

@app . route ('/search' , methods = ['GET']) def

search (): query =

"I want a tarot reading" available_ais = fetch . ai (query) sender_identity = Identity . from_seed ("whatever i want this to be, but i am searching" , 0) for ai in available_ais . get ('ais'): payload =

{ "date of birth" :

"around the same time as dinosaurs" , "gender" :

"reptile" }

other_addr

ai . get ("address" , "")

print (f "sending a message to an ai, { other_addr } ")

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

return

{ "status" :

"Agent searched" }

@app . route ('/webhook' , methods = ['POST']) def

webhook (): data = request . json print (f "webhook { data } ") try : message =

parse_message_from_agent (json. dumps (data)) except

ValueError

as e : print (f " { e } ") return

{ "status" :

f "error: { e } " }

payload

message . payload

print (payload)

return

{ "status" :

"Agent message processed" }

if

name

==

"**main**" : app . run (host = '0.0.0.0' , port = 5002 , debug = True) The main difference in the search Agent is that we define a/search endpoint . This is very interesting, withfetch.ai(query) we can find any Agent in themarketplace . Then we can send a message to all of them asking for a tarot reading. The search is a combination of Elastic search and multi-layered LLMs, meaning natural language queries can find you accurate results.

@app.route('/search', methods=['GET']) def search(): query = "I want a tarot reading" available_ais = fetch.ai(query) for ai in available_ais.get('ais'): send_message_to_agent(...)

Running the agents.

We run these agents with the following commands, they will need to be run in separate terminals.

Run the following commands in each terminal window:

python tarot_agent.py and

python tarot_search_agent.py Now, they're running. Let's get them registered. In a new terminal window:

curl --location '127.0.0.1:5000/register' curl --location '127.0.0.1:5002/register' Then, you can send another GET request to thetarot_search_agent , and it should then go and find you an Agent which will return a tarot reading.

Let's open a terminal window and curl ourtarot_search_agent 's IP to start the search process:

curl --location '127.0.0.1:5002/search'

Expected output

The expected output from thetarot_search_agent should be similar to the following:

Serving

Flask

app

'tarot_search' *

Debug

mode:

on WARNING:

This is a development server. Do not use it in a production deployment. Use a production WSGI server instead. * Running on all addresses (0.0.0.0) * Running on http://127.0.0.1:5002 Press CTRL+C to quit * Restarting with stat * Debugger is active! * Debugger PIN:

751 -296-284 sending

a

message

to

an

ai,

agent1qwpd8cy9ymhjyuj4x2k75dv69vlxquk0xtwhmw09khv8jdkszw32y7rfd99 … { 'date of birth' :

'around the same time as dinosaurs' ,

'gender' :

'reptile' } …. { 'Response' :

"Certainly! Let's dive into your tarot reading, my ancient reptilian friend.\n\n ..." } Last updated on January 20, 2025

Was this page helpful?

You can also leave detailed feedbackon Github

AI Engine Javascript SDK Startup Idea analyzer

On This Page