How to use on_query decorator

Introduction

on_query decorator in uagents is used to register a function as a handler for incoming queries that match a specified model. These decorators enable the agent to respond to queries in an event-driven manner.

Walk - through

1. First of all naviage to directory where you want to make your project.
2. Create a python script nameon_query.py
3. usingtouch on_query.py
4. .
5. We need to importjson, asyncio, uagent's model and query
6. . Then we would need to define the query format using the QueryRequest class as a subclass of Model:

import required libraries

import json import asyncio from uagents import Model from uagents . query import query

Add your agent's address

AGENT_ADDRESS

"Agent Address"

class

QueryRequest (Model): query :

str 1. Createagent_query 2. function to send query to agent and decode the response recieved.

async

def

agent_query (req): response =

await

query (destination = AGENT_ADDRESS, message = req, timeout = 15.0) data = json . loads (response. decode_payload ()) return data ["text"] 1. Createmake_agent_call 2. function to handle query responses by the agent.

async

def

make_agent_call (req : QueryRequest): try : response =

await

agent_query (req) return

f "successful call - agent response: { response } " except

Exception : return

"unsuccessful agent call" 1. Finally, to run the script and send a query to the agent, initialize a QueryRequest object with your query message, then call make_agent_call within an asyncio event loop. This will send the query to the specified agent address and print the response, indicating whether the call was successful or not. This completes the setup for using the on_query decorator in a uAgents project to handle incoming queries.

if

name

==

"**main**" : request =

QueryRequest (message = "Your query here") print (asyncio. run (make_agent_call (request))).

On_query Script

on_query.py

Importing required libraries

import json import asyncio from uagents import Model from uagents . query import query

Define the agent's address to send queries to.

AGENT_ADDRESS

"Agent Address"

Define a model for the query request.

class

QueryRequest (Model): query :

str

Asynchronous function to send a query to the specified agent.

async

def

agent_query (req): response =

await

query (destination = AGENT_ADDRESS, message = req, timeout = 15.0) data = json . loads (response. decode_payload ())

Decode the payload from the response and load it as JSON.

return data ["text"]

Asynchronous function to make a call to an agent and handle the response.

async

def

make_agent_call (req : QueryRequest): try : response =

await

agent_query (req) return

f "successful call - agent response: { response } " except

Exception : return

"unsuccessful agent call"

Main block to execute the script.

if

name

==

"**main**" :

Create a QueryRequest instance with your query and run make_agent_call with request.

request

QueryRequest (message = "Your query here") print (asyncio. run (make_agent_call (request)))

Agent's Script.

For the agent section, the script sets up a uAgents-based agent to handle incoming queries. It defines two models:QueryRequest for incoming queries andResponse for replies. Upon startup, it logs the agent's details. The core functionality lies in thequery_handler , decorated with @QueryAgent.on_query, which processes received queries and sends back a predefined response. This demonstrates creating responsive agents within the uAgents framework, showcasing how they can interact with other agents or services in an asynchronous, event-driven architecture.

agent.py from uagents import Agent , Context , Model

Define the request and response model.

class

QueryRequest (Model): message :

str

The query message.

class

Response (Model): text :

str

The response text.

Initialize the agent with its configuration.

QueryAgent

Agent (name = "Query Agent" , seed = "Query Agent Seed Phrase" , port = 8001 , endpoint = "http://localhost:8001/submit" ,)

Getting agent details on startup

@QueryAgent . on_event ("startup") async

def

startup (ctx : Context): ctx . logger . info (f "Starting up { QueryAgent.name } ") ctx . logger . info (f "With address: { QueryAgent.address } ") ctx . logger . info (f "And wallet address: { QueryAgent.wallet. address () } ")

Decorator to handle incoming queries.

@QueryAgent . on_query (model = QueryRequest, replies = {Response}) async

def

query_handler (ctx : Context ,

sender :

str ,

_query : QueryRequest): ctx . logger . info ("Query received")

Log receipt of query.

try : await ctx . send (sender, Response (text = "success")) except

Exception : await ctx . send (sender, Response (text = "fail"))

Main execution block to run the agent.

if

name

==

"**main**" : QueryAgent . run ()

Expected Output

- Agent.py

INFO: [Query Agent]: Almanac registration is up to date! INFO: [Query Agent]: Starting up Query Agent INFO: [Query Agent]: With address: agent1qgfytc9e7ketwqc06xndvjmznqgr3md8w43hzxdv2hasp25ya43j2mnd32e INFO: [Query Agent]: And wallet address: fetch1qlq2nnegdj3axk7ms3qgrez7l6032s2k9s7704 INFO: [Query Agent]: Starting server on http://0.0.0.0:8001 (Press CTRL+C to quit) INFO: [Query Agent]: Query received * on_query.py

successful call - agent response: success

Was this page helpful?

Running a Locally Hosted Agent with LangChain Integration Running Locally