How to use agents to send tokens

Introduction

Within agent-based decentralized systems, efficient communication and secure data exchange are essential. In this guide, we will walk you through the process of setting up two AI Agents utilizing theuagents library to establish a dynamic workflow where one agent periodically sends payment requests to another, which in turn processes these requests, executes transactions, and provides transaction information back to sending agent.

Let's get started!

Walk-through

1. First of all, create a Python script for this task, and name it by running:touch sending_tokens.py
2. Then, import the necessary modules fromuagents
3. ,uagents.network
4. , anduagents.setup
5. . Let's then define two data models:PaymentRequest
6. andTransactionInfo
7. . We then need to set up the values for theAMOUNT
8. andDENOM
9. variables, which define the default amount and denomination for the payment requests:
10. from
11. uagents
12. import
13. Agent
14. ,
15. Bureau
16. ,
17. Context
18. ,
19. Model
20. from
21. uagents
22. .
23. network
24. import
25. wait_for_tx_to_complete
26. from
27. uagents
28. .
29. setup
30. import
31. fund_agent_if_low
32. class
33. PaymentRequest
34. (
35. Model
36.):
37. wallet_address
38. :
39. str
40. amount
41. :
42. int
43. denom
44. :
45. str
46. class
47. TransactionInfo
48. (
49. Model
50.):
51. tx_hash
52. :
53. str

AMOUNT
=
100
DENOM
=
"atestfet"

- ThePaymentRequest
- model represents a payment request which contains thewallet_address
- ,amount
- , anddenom
- .
- TheTransactionInfo
- model represents information about a transaction and contains a single attribute,tx_hash
- .

Let's now define our agents,alice
andbob
. Ensure they have enough funds in their wallets to carry out transactions:
alice
=
Agent
(name
=
"alice"
, seed
=
"alice secret phrase"
)
bob
=
Agent
(name
=
"bob"
, seed
=
"bob secret phrase"
)
fund_agent_if_low
(bob.wallet.
address
())
fund_agent_if_low
(alice.wallet.
address
())
We can now define our agents behaviour and functions:
@alice
.
on_interval
(period
=
10.0
)
async
def
request_funds
(
ctx
:
Context):

114. await
115. ctx
116. .
117. send
118. (
119. bob.address,
120. PaymentRequest
121. (
122. wallet_address
123. =
124. str
125. (ctx.wallet.
126. address
127. ()), amount
128. =
129. AMOUNT, denom
130. =
131. DENOM
132.),
133.)
134. This defines an event handler foralice
135. using the.on_interval()
136. decorator. This event handler is triggered at regular intervals of10.0
137. seconds in this case. The event handler function is namedrequest_funds()
138. and takes actx
139. parameter of typeContext
140. . Within the function,alice
141. sends a payment request message tobob
142. by using thectx.send()
143. method. Thectx.send()
144. method is called with the recipient address,bob.address
145. , which specifies that the message should be sent tobob
146. . The message is an instance of thePaymentRequest()
147. model. It containsalice
148. 's wallet address (ctx.wallet.address()
149.), the amount (AMOUNT
150.), and the denomination (DENOM
151.).
152. We can now define aconfirm_transaction()
153. message handler foralice
154. to handle incoming messages frombob
155. of typeTransactionInfo
156. :
157. @alice
158. .
159. on_message
160. (model
161. =
162. TransactionInfo)
163. async
164. def
165. confirm_transaction
166. (
167. ctx
168. :
169. Context
170. ,
171. sender
172. :
173. str
174. ,
175. msg
176. :
177. TransactionInfo):
178. ctx
179. .
180. logger
181. .

```
info
(
f
"Received transaction info from
{
sender
}
:
{
msg
}
"
)
tx_resp
=
await
wait_for_tx_to_complete
(msg.tx_hash, ctx.ledger)
coin_received
=
tx_resp
.
events
[
"coin_received"
]
if
(
coin_received
[
"receiver"
]
==
str
(ctx.wallet.
address
())
and
coin_received
[
"amount"
]
==
f
"
{
AMOUNT
}{
DENOM
}
"
)
:
ctx
.
logger
.
info
(
f
"Transaction was successful:
{
coin_received
}
"
)
The event handler function is namedconfirm_transaction()
and takes three parameters:ctx
```

,sender
, andmsg
. Within the function,alice
logs an informational message using thectx.logger.info()
method, indicating the receipt of transaction information from the sender agent,bob
, and displaying themsg
object. Thewait_for_tx_to_complete()
function is used to await the completion of the transaction specified by thetx_hash
received in the message.
Once the transaction is completed, the code accesses thecoin_received
event from the transaction response usingtx_resp.events[\"coin_received\"]
. It checks if the receiver address matchesalice
's wallet address (ctx.wallet.address()
) and if the amount received matches the expected amount (AMOUNT + DENOM
).
If the conditions are met,alice
logs another informational message indicating the success of the transaction and displaying the details of the received coins.
Let's now define an event handler forbob
. This event handler is triggered whenbob
receives a message of typePaymentRequest
fromalice
:
@bob
.
on_message
(model
=
PaymentRequest, replies
=
TransactionInfo)
async
def
send_payment
(
ctx
:
Context
,
sender
:
str
,
msg
:
PaymentRequest):
ctx
.
logger
.
info
(
f
"Received payment request from
{
sender
}
:
{
msg
}
"
)

send the payment

transaction

=
ctx
.
ledger
.
send_tokens
(
msg.wallet_address, msg.amount, msg.denom, ctx.wallet
)

send the tx hash so alice can confirm

await
ctx
.
send
(alice.address,
TransactionInfo
(tx_hash
=
transaction.tx_hash))
The event handler function is namedsend_payment()
and takes three parameters:ctx
,sender
, andmsg
. Within the function,bob
logs an informational message using thectx.logger.info()
method, indicating the receipt of a payment request from the sender agent,bob
, and displaying themsg
object.
Next, the code performs a payment transaction using thectx.ledger.send_tokens()
method. It takes the wallet address (msg.wallet_address
), amount (msg.amount
), denomination (msg.denom
), andctx.wallet()
as parameters. This method is responsible for sending the requested payment.
Once the transaction is completed,bob
sends a message back toalice
to inform her about the transaction, usingctx.send()
. The message is created using theTransactionInfo
model with thetx_hash
obtained from the transaction response. Thectx.send()
method is used to send this message to alice using her address (alice.address
).
We are now ready to use the Bureau class to create abureau
object and add both uAgents to it so for them to be run together:
bureau
=
Bureau
()
bureau
.
add
(alice)
bureau
.
add
(bob)
if
name
==
"**main**"
:
bureau
.
run
()

The overall script for this example should look as follows:

sending_tokens.py from uagents import Agent , Bureau , Context , Model from uagents . network import wait_for_tx_to_complete from uagents . setup import fund_agent_if_low

class

PaymentRequest (Model): wallet_address :

str amount :

int denom :

str

class

TransactionInfo (Model): tx_hash :

str

AMOUNT

100 DENOM =

"atestfet"

alice

Agent (name = "alice" , seed = "alice secret phrase") bob =

Agent (name = "bob" , seed = "bob secret phrase")

fund_agent_if_low (bob.wallet. address ()) fund_agent_if_low (alice.wallet. address ())

@alice . on_interval (period = 10.0) async

def

request_funds (ctx : Context): await ctx . send (bob.address, PaymentRequest (wallet_address = str (ctx.wallet. address ()), amount = AMOUNT, denom = DENOM),)

@alice . on_message (model = TransactionInfo) async

def

confirm_transaction (ctx : Context ,

sender :

str ,

msg : TransactionInfo): ctx . logger . info (f "Received transaction info from { sender } : { msg } ") tx_resp =

await

wait_for_tx_to_complete (msg.tx_hash, ctx.ledger)

coin_received

tx_resp . events ["coin_received"] if (coin_received ["receiver"]

==

str (ctx.wallet. address ()) and coin_received ["amount"]

==

f " { AMOUNT }{ DENOM } ") : ctx . logger . info (f "Transaction was successful: { coin_received } ")

@bob . on_message (model = PaymentRequest, replies = TransactionInfo) async

def

send_payment (ctx : Context ,

sender :

str ,

msg : PaymentRequest): ctx . logger . info (f "Received payment request from { sender } : { msg } ")

send the payment

transaction

ctx . ledger . send_tokens (msg.wallet_address, msg.amount, msg.denom, ctx.wallet)

send the tx hash so alice can confirm

await ctx . send (alice.address, TransactionInfo (tx_hash = transaction.tx_hash))

bureau

Bureau () bureau . add (alice) bureau . add (bob)

if

name

==

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

Run the script

On your terminal, make sure to have activated the virtual environment.

Run the script:python sending_tokens.py

The output should be as follows:

[bob]: Received payment request from agent1qdp9j2ev86k3h5acaayjm8tpx36zv4mjxn05pa2kwesspstzj697xy5vk2a: wallet_address='fetch1967p3vkp0yngdfturv4ypq2p4g760ml705wcxy' amount=100 denom='atestfet' [alice]: Received transaction info from agent1q2kxet3vh0scsf0sm7y2erzz33cve6tv5uk63x64upw5g68kr0chkv7hw50: tx_hash='DB662CCF415C7B0C9A02928967BE1817506D02A041AA05CA48DCE5CF87D5A638' [alice]: Transaction was successful: {'receiver': 'fetch1967p3vkp0yngdfturv4ypq2p4g760ml705wcxy', 'amount': '100atestfet'}

Was this page helpful?

How to use agents to verify messages How to use the agents to simulate a cleaning scenario ✨