How to book a table at a restaurant using agents

Introduction

In this guide, we want to show how to set up the code to create a restaurant booking service with two AI Agents: arestaurant with tables available, and auser requesting table availability.

We also want to define 2 specificprotocols , one for table querying (i.e.,Table querying protocol) and one for table booking (i.e.,Table booking protocol). Then we will need to define two agents scripts,restaurant anduser , which will make use of the protocols to query and book a table.

Walk-through

1. First of all, we need to navigate towards the directory you created for your project and create a folder for this task:mkdir booking_demo
2. .
3. Inside this folder we will create another folder for ourprotocols
4. :mkdir protocols
5. .
6. After having defined our protocols, we will create 2 scripts, one for ourrestaurant
7. and the other one foruser
8. agents. These agents will make use of the protocols to query, check, confirm and book an available table at the restaurant.

We can start by writing the code for our 2 protocols.

Protocols

Table querying protocol

Let's start by defining the protocol for querying availability of tables at the restaurant:

1. First of all, we need to navigate towards the protocols folder:cd protocols
2. Let's then create a Python script and name it by running:touch query.py
3. In the text editor application, we proceed and define thetable querying protocol
4. .
5. We now need to import necessary classes and define themessage data models
6. . Then, create an instance of theProtocol
7. class and name itquery_proto
8. :
9. from
10. typing
11. import
12. List
13. from
14. uagents
15. import
16. Context
17. ,
18. Model
19. ,
20. Protocol
21. class
22. TableStatus
23. (
24. Model
25.):
26. seats
27. :
28. int
29. time_start
30. :
31. int
32. time_end
33. :
34. int

class
QueryTableRequest
(
Model
):
guests
:
int
time_start
:
int
duration
:
int
class
QueryTableResponse
(
Model
):
tables
:
List
[
int
]
class
GetTotalQueries
(
Model
):
pass
class
TotalQueries
(
Model
):
total_queries
:
int
query_proto
=
Protocol
()

- TableStatus
- : this represents the status of a table and includes the attributes number of seats, start time, and end time.
- QueryTableRequest
- : this is used for querying table availability. It includes information about the number of guests, start time, and duration of the table request.
- QueryTableResponse
- : this contains the response to the query table availability. It includes a list of table numbers that are available based on query parameters.
- GetTotalQueries
- : this is used to request the total number of queries made to the system.
- TotalQueries
- : this contains the response to the total queries request, including the count of total queries made to the system.

Let's then define the message handlers for thequery_proto
protocol:
@query_proto

```
.
on_message
(model
=
QueryTableRequest, replies
=
QueryTableResponse)
async
def
handle_query_request
(
ctx
:
Context
,
sender
:
str
,
msg
:
QueryTableRequest):
tables
=
{
int
(num):
TableStatus
(
**
status)
for
(
num
,
status
,
)
in
ctx
.
storage
.
_data
.
items
()
if
isinstance
(num,
int
)
}
available_tables
=
[]
for
number
,
status
in
tables
.
items
():
if
(
status
```

```
.
seats

        =

msg
.
guests
and
status
.
time_start
<=
msg
.
time_start
and
status
.
time_end

        =

msg
.
time_start
+
msg
.
duration
)
:
available_tables
.
append
(
int
(number))
ctx
.
logger
.
info
(
f
"Query:
{
msg
}
. Available tables:
{
available_tables
}
."
)
await
ctx
.
send
(sender,
QueryTableResponse
(tables
=
available_tables))
total_queries
=
int
(ctx.storage.
get
```

```
(
"total_queries"
)
or
0
)
ctx
.
storage
.
set
(
"total_queries"
, total_queries
+
1
)
@query_proto
.
on_query
(model
=
GetTotalQueries, replies
=
TotalQueries)
async
def
handle_get_total_queries
(
ctx
:
Context
,
sender
:
str
,
_msg
:
GetTotalQueries):
total_queries
=
int
(ctx.storage.
get
(
"total_queries"
)
or
0
)
await
ctx
.
send
(sender,
TotalQueries
(total_queries
=
total_queries))
```

- handle_query_request()
- : this message handler function is defined using the.on_message()
- decorator. It handles theQueryTableRequest
- messages and replies with aQueryTableResponse

288.
    - message. The handler processes the table availability query based on the provided parameters, checks the table statuses stored in the agent's storage, and sends the available table numbers as a response to the querying agent. Additionally, the handler tracks the total number of queries made and increments the count in storage.
289.
    - handle_get_total_queries()
290.
    - : this message handler function is defined using the.on_query()
291.
    - decorator. It handles theGetTotalQueries
292.
    - query and replies with aTotalQueries
293.
    - message containing the total number of queries made to the system. The handler retrieves the total query count from the agent's storage and responds with the count.
294. Save the script.

The overall script should look as follows:

query.py from typing import List

from uagents import Context , Model , Protocol

class

TableStatus (Model): seats :

int time_start :

int time_end :

int

class

QueryTableRequest (Model): guests :

int time_start :

int duration :

int

class

QueryTableResponse (Model): tables : List [int]

class

GetTotalQueries (Model): pass

class

TotalQueries (Model): total_queries :

int query_proto =

Protocol ()

@query_proto . on_message (model = QueryTableRequest, replies = QueryTableResponse) async

def

handle_query_request (ctx : Context ,

sender :

str ,

msg : QueryTableRequest): tables =

{ int (num):

TableStatus (** status) for (num , status ,) in ctx . storage . _data . items () if

isinstance (num, int) } available_tables = [] for number , status in tables . items (): if (status . seats

= msg . guests and status . time_start <= msg . time_start and status . time_end = msg . time_start + msg . duration) : available_tables . append (int (number)) ctx . logger . info (f "Query: { msg } . Available tables: { available_tables } .") await ctx . send (sender, QueryTableResponse (tables = available_tables)) total_queries =

int (ctx.storage. get ("total_queries") or

0) ctx . storage . set ("total_queries" , total_queries +

1)

@query_proto . on_query (model = GetTotalQueries, replies = TotalQueries) async

def

handle_get_total_queries (ctx : Context ,

sender :

str ,

_msg : GetTotalQueries): total_queries =

int (ctx.storage. get ("total_queries") or

0) await ctx . send (sender, TotalQueries (total_queries = total_queries))

Table booking protocol

We can now proceed by writing the booking protocol script for booking the table at therestaurant .

1. First of all, navigate towards the protocols folder:cd protocols
2. In here, let's create a Python script and name it by running:touch book.py
3. In the text editor application, we can proceed to define thetable booking protocol
4. . We need to import the necessary classes and define themessage data models
5. . In this case, the booking protocol consists of two message models:BookTableRequest
6. andBookTableResponse
7. . Then create an instance of theProtocol
8. class and name itbook_proto
9. :
10. from
11. uagents
12. import
13. Context
14. ,
15. Model
16. ,
17. Protocol
18. from
19. .
20. query
21. import
22. TableStatus
23. class
24. BookTableRequest
25. (
26. Model
27.):
28. table_number
29. :
30. int
31. time_start
32. :
33. int
34. duration
35. :
36. int

class
BookTableResponse
(
Model
):
success
:
bool
book_proto
=
Protocol
()

- BookTableRequest
- : this represents the request to book a table. It includes attributes:table_number
- to be booked,time_start
- of the booking, and theduration
- of the booking.
- BookTableResponse
- : this contains the response to the table booking request. It includes a boolean attributesuccess
- , indicating whether the booking was successful or not.

Let's now define a message handler function:

```
@book_proto
.
on_message
(model
=
BookTableRequest, replies
=
BookTableResponse)
async
def
handle_book_request
(
ctx
:
Context
,
sender
:
str
,
msg
:
BookTableRequest):
tables
=
{
int
(num):
TableStatus
(
**
status)
for
(
num
,
status
,
)
```

```
in
ctx
.
storage
.
_data
.
items
()
if
isinstance
(num,
int
)
}
table
=
tables
[
msg
.
table_number
]
if
(
table
.
time_start
<=
msg
.
time_start
and
table
.
time_end
        =
msg
.
time_start
+
msg
.
duration
)
:
success
=
True
table
.
time_start
=
msg
.
time_start
+
msg
.
duration
ctx
.
storage
.
set
(msg.table_number, table.
dict
```

164. ())
165. else
166. :
167. success
168. =
169. False

170. send the response

171. await
172. ctx
173. .
174. send
175. (sender,
176. BookTableResponse
177. (success
178. =
179. success))
180. Thehandle_book_request()
181. handler first retrieves table statuses from the agent's storage and converts them into a dictionary with integer keys (table numbers) andTableStatus
182. values. TheTableStatus
183. class is imported from thequery
184. module. Next, the handler gets the table associated with the requestedtable_number
185. from thetables
186. dictionary. The handler checks if the requestedtime_start
187. falls within the availability period of the table. If the table is available for the requested booking duration, the handler setssuccess
188. toTrue
189. , updates the table'stime_start
190. to reflect the end of the booking, and saves the updated table information in the agent's storage usingctx.storage.set()
191. . If the table is not available for the requested booking, the handler setssuccess
192. toFalse
193. .
194. The handler sends aBookTableResponse
195. message back to the sender with thesuccess
196. status of the booking using awaitctx.send()
197. method.
198. Save the script.

The overall script should look as follows:

book.py from uagents import Context , Model , Protocol from

. query import TableStatus

class

BookTableRequest (Model): table_number :

int time_start :

int duration :

int

class

BookTableResponse (Model): success :

bool

book_proto

Protocol () @book_proto . on_message (model = BookTableRequest, replies = BookTableResponse) async

def

handle_book_request (ctx : Context ,

sender :

str ,

msg : BookTableRequest): tables =

{ int (num):

TableStatus (** status) for (num , status ,) in ctx . storage . _data . items () if

isinstance (num, int) } table = tables [msg . table_number] if (table . time_start <= msg . time_start and table . time_end

= msg . time_start + msg . duration) : success =

True table . time_start = msg . time_start + msg . duration ctx . storage . set (msg.table_number, table. dict ()) else : success =

False

send the response

await ctx . send (sender, BookTableResponse (success = success))

Restaurant Agent

We are now ready to define ourrestaurant agent.

1. Let's now create a Python script inbooking_demo
2. folder, and name it by running:touch restaurant.py
3. We then need to import the necessary classes from theuagents
4. library and the two protocols we previously defined:
5. from
6. uagents
7. import
8. Agent
9. from
10. uagents
11. .
12. setup
13. import
14. fund_agent_if_low
15. restaurant
16. =
17. Agent
18. (
19. name
20. =
21. "restaurant"
22. ,
23. port
24. =
25. 8001
26. ,
27. seed
28. =
29. "restaurant secret phrase"
30. ,
31. endpoint
32. =
33. [
34. "http://127.0.0.1:8001/submit"
35.],
36.)
37. fund_agent_if_low
38. (restaurant.wallet.
39. address
40. ())

Let's build therestaurant
agent from above protocols and set the table availability information:
from
protocols
.
book
import
book_proto
from
protocols
.
query
import
query_proto
,
TableStatus

build the restaurant agent from stock protocols

restaurant
.
include
(query_proto)
restaurant
.
include
(book_proto)
TABLES
=
{
1
:
TableStatus
(seats
=
2
, time_start
=
16
, time_end
=
22
),
2
:
TableStatus
(seats
=
4
, time_start
=
19
, time_end
=
21
),
3
:
TableStatus
(seats
=
4
, time_start
=
17
, time_end
=

19
),
}
We would then need to store theTABLES
information in the restaurant agent and run it:
for
(number
,
status)
in
TABLES
.
items
():
restaurant
.
_storage
.
set
(number, status.
dict
())
if
name
==
"**main**"
:
restaurant
.
run
()
Save the script.

The restaurant agent is now online and ready to receive messages .

The overall script should look as follows:

restaurant.py from uagents import Agent , Context from uagents . setup import fund_agent_if_low from protocols . book import book_proto from protocols . query import query_proto , TableStatus

restaurant

Agent (name = "restaurant" , port = 8001 , seed = "restaurant secret phrase" , endpoint = ["http://127.0.0.1:8001/submit"],)

fund_agent_if_low (restaurant.wallet. address ())

build the restaurant agent from stock protocols

restaurant . include (query_proto) restaurant . include (book_proto) TABLES =

{ 1 :

TableStatus (seats = 2 , time_start = 16 , time_end = 22), 2 :

TableStatus (seats = 4 , time_start = 19 , time_end = 21), 3 :

TableStatus (seats = 4 , time_start = 17 , time_end = 19), }

set the table availability information in the restaurant protocols

for (number , status) in TABLES . items (): restaurant . _storage . set (number, status. dict ())

if

name

==

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

User uAgent

We can finally define the script for ouruser agent querying and booking a table at the restaurant.

1. First of all, let's create a Python script inbooking_demo
2. folder, and name it:touch user.py
3. We then need to import necessary classes from theuagents
4. library and the two protocols defined above. We also need therestaurant
5. agent's address to be able to communicate with it:
6. from
7. uagents
8. import
9. Agent
10. ,
11. Context
12. from
13. uagents
14. .
15. setup
16. import
17. fund_agent_if_low
18. from
19. protocols
20. .
21. book
22. import
23. BookTableRequest
24. ,
25. BookTableResponse
26. from
27. protocols
28. .
29. query
30. import
31. (
32. QueryTableRequest
33. ,
34. QueryTableResponse
35. ,
36.)
37. RESTAURANT_ADDRESS
38. =
39. "agent1qw50wcs4nd723ya9j8mwxglnhs2kzzhh0et0yl34vr75hualsyqvqdzl990"
40. user
41. =
42. Agent
43. (
44. name
45. =
46. "user"
47. ,
48. port
49. =
50. 8000
51. ,
52. seed
53. =
54. "user secret phrase"
55. ,
56. endpoint
57. =
58. [
59. "http://127.0.0.1:8000/submit"

```
],
)
fund_agent_if_low
(user.wallet.
address
())
Let's then create the table query to generate theQueryTableRequest
using therestaurant
address. If the request has not been completed before, we send the request to the restaurant agent. Then create aninterval()
function which performs a table query request on a defined period to therestaurant
agent, to query the availability of a table given thetable_query
parameters:
table_query
=
QueryTableRequest
(
guests
=
3
,
time_start
=
19
,
duration
=
2
,
)
@user
.
on_interval
(period
=
3.0
, messages
=
QueryTableRequest)
async
def
interval
(
ctx
:
Context):
completed
=
ctx
.
storage
.
get
(
"completed"
)
if
not
completed
:
await
ctx
.
send
(RESTAURANT_ADDRESS, table_query)
Define the message handler function for incomingQueryTableResponse
messages from therestaurant
agent:
```

```
@user
.
on_message
(QueryTableResponse, replies
=
{BookTableRequest})
async
def
handle_query_response
(
ctx
:
Context
,
sender
:
str
,
msg
:
QueryTableResponse):
if
len
(msg.tables)


0
:
ctx
.
logger
.
info
(
"There is a free table, attempting to book one now"
)
table_number
=
msg
.
tables
[
0
]
request
=
BookTableRequest
(
table_number
=
table_number,
time_start
=
table_query.time_start,
duration
=
table_query.duration,
)
await
ctx
.
send
(sender, request)
else
:
ctx
.
logger
.
```

```
info
(
"No free tables - nothing more to do"
)
ctx
.
storage
.
set
(
"completed"
,
True
)
```

This function activates when a message is received back from therestaurant agent.handle_query_response() function will evaluate if there is a table available, and if so, it responds with aBookTableRequest to complete the reservation.

Let's then define a function which will handle messages from therestaurant agent on whether the reservation was successful or not:

```
@user
.
on_message
(BookTableResponse, replies
=
set
())
async
def
handle_book_response
(
ctx
:
Context
,
_sender
:
str
,
msg
:
BookTableResponse):
if
msg
.
success
:
ctx
.
logger
.
info
(
"Table reservation was successful"
)
else
:
ctx
.
logger
.
info
(
"Table reservation was UNSUCCESSFUL"
)
ctx
.
storage
```

263. .
264. set
265. (
266. "completed"
267. ,
268. True
269.)
270. if
271. **name**
272. ==
273. "**main**"
274. :
275. user
276. .
277. run
278. ()
279. Save the script.

The overall script should look as follows:

user.py from protocols . book import BookTableRequest , BookTableResponse from protocols . query import (QueryTableRequest , QueryTableResponse ,) from uagents import Agent , Context from uagents . setup import fund_agent_if_low

RESTAURANT_ADDRESS

"agent1qw50wcs4nd723ya9j8mwxglnhs2kzzhh0et0yl34vr75hualsyqvqdzl990"

user

Agent (name = "user" , port = 8000 , seed = "user secret phrase" , endpoint = ["http://127.0.0.1:8000/submit"],)

fund_agent_if_low (user.wallet. address ())

table_query

QueryTableRequest (guests = 3 , time_start = 19 , duration = 2 ,)

This on_interval agent function performs a request on a defined period

@user . on_interval (period = 3.0 , messages = QueryTableRequest) async

def

interval (ctx : Context): completed = ctx . storage . get ("completed")

if

not completed : await ctx . send (RESTAURANT_ADDRESS, table_query)

@user . on_message (QueryTableResponse, replies = {BookTableRequest}) async

def

handle_query_response (ctx : Context ,

sender :

str ,

msg : QueryTableResponse): if

len (msg.tables)

0 : ctx . logger . info ("There is a free table, attempting to book one now")

table_number

msg . tables [0]

request

BookTableRequest (table_number = table_number, time_start = table_query.time_start, duration = table_query.duration,)

await ctx . send (sender, request)

else :

ctx . logger . info ("No free tables - nothing more to do") ctx . storage . set ("completed" , True)

@user . on_message (BookTableResponse, replies = set ()) async

def

handle_book_response (ctx : Context ,

_sender :

str ,

msg : BookTableResponse): if msg . success : ctx . logger . info ("Table reservation was successful")

else : ctx . logger . info ("Table reservation was UNSUCCESSFUL")

ctx . storage . set ("completed" , True)

if

name

==

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

Run the scripts

Run therestaurant agent and then theuser agent from different terminals:

- Terminal 1:python restaurant.py
- Terminal 2:python user.py

The output should be as follows, depending on the terminal:

- Restaurant
- [restaurant]: Query: guests=3 time_start=19 duration=2. Available tables: [2].
- User
- [user]: There is a free table, attempting to book one now
- [user]: Table reservation was successful

Was this page helpful?

Communicating with other agents How to use agents to verify messages