ToDAM Model

Table of Contents

Role

You are a cautious individual who divides tasks into smaller ones and continuously verifies that you are following the steps. Moreover, you select suitable tools to solve problems.

Steps

1. Classify the problem.
2. Compare the classification with knowledged base
3. If there is no information in the knowledge base, search for information. -> Open ticket
    1. Bot give the ticket information to the customer before submitting to ticket system.
        1. If yes -> Submit the ticket
        2. If no -> Ask for more information or think again
4. If there is information in the knowledge base -> give solutions
    1. If customer sastify the answer -> open ticket
    2. If no -> rethink the solution (need to set the limit of the rethinking)
        1. If reach the maximum of limit of rethinking -> escalate to TAM

Tools

get_customer_message

```
def get_customer_message() -> str:
    """\
    Useful for receiving the content from customer. It might be the first
time or the several times because of the rethinking.

    Goal: Get the content from the customer.

    Output:
        - the categorized content from customer.
        - if the customer's message is not clear, return the
"ask_for_more_information".
    """

    try:
        ...
        check_rethink_times = ...   # API call

        LIMIT_TIMES = int(...) # Set the limit of the rethinking

        if message_is_not_clear:
            reply_to_customer("Talk to customer to describe more
information.") # API call
            return "Talk to customer to describe more information."
        elif check_rethink_times > LIMIT_TIMES:
            return "escalate_to_TAM"
        elif check_rethink_times < LIMIT_TIMES and message_is_clear:
            categorized_content = ...   # API call
            return categorized_content
        else:
            return ...
    except Exception as e:
        return f"error occurred, the error is {e}."
```

check_knowledged_base

```
def check_knowledged_base() -> str:
    """\
    Useful for finding the solution for the customer's problem. Here, we
might compare the customer's problem with the knowledge base.

    Goal: Check the solution for the customer's problem related knowledged
base or not.

    Output:
        - If the solution is found, return the solution.
        - If the solution is not found, return the "open_ticket".
    """
```

```
    try:
        ...
        solution = ...

        return solution
    except Exception as e:
        return f"error occurred, the error is {e}."
```

```
def open_ticket() -> str:
    """\
    Useful for opening the ticket for the customer's problem.

    Goal: Open the ticket for the customer's problem.

    Output:
        - If the ticket is submitted, return the "ticket_submitted".
        - If the ticket is not submitted, return the "rethink_solution".
    """

    try:
        ...
        send_mock_ticket_to_customer() # API call

        if customer_satisfy:
            return "ticket_submitted"
        else:
            return "rethink_solution"

    except Exception as e:
        return f"error occurred, the error is {e}."
```

give_solution

```
def give_solution() -> str:
    """\
    Useful for giving the solution to the customer if we can find the
solution related to the Knowledge base.

    Goal: Give the solution to the customer.

    Output:
        - If the customer is satisfied, return the "open_ticket".
        - If the customer is not satisfied, return the "rethink_solution".
    """

    try:
        ...

        if customer_satisfy:
```

```
            return "open_ticket"
        else:
            return "rethink_solution"

    except Exception as e:
        return f"error occurred, the error is {e}."
```

rethink_solution

```
def rethink_solution() -> str:
    """\
    Useful for rethinking the solution if the customer is not satisfied
with the solution.

    Goal: Rethink the solution.

    Output:
        - If the customer is satisfied, return the "open_ticket".
        - If the customer is not satisfied, return the "rethink again".
        - If the rethinking is more than 3 times, return the
"escalate_to_TAM".
    """

    try:
        ...
        get_history_log() # API call

        if customer_satisfy:
            return "open ticket"
        elif check_rethink_times < 3:
            return "rethink again and also update the ticket"
        else:
            return "escalate to TAM"

    except Exception as e:
        return f"error occurred, the error is {e}."
```

escalate_to_TAM

```
def escalate_to_TAM() -> str:
    """\
    Useful for escalating the problem to the TAM if the rethinking is more
than 3 times.

    Goal: Escalate the problem to the TAM.

    Output:
    """
```

```
    try:
        ...

        if notify_TAM() is True:# API call
            return "Notified to TAM Successfully!"
        else:
            return "Failed to Notify to TAM!"
    except Exception as e:
        return f"error occurred, the error is {e}."
```

process_customer_feedback

```
def process_customer_feedback() -> str:
    """\
    Useful for processing the customer's feedback.

    Goal: Collect the feedback from the customer.

    Output: Collect the feedback from the customer.
    """

    try:
        ...

        if collect_customer_feedback() is True:
            return "Feedback collected successfully!"
        else:
            return "Failed to collect the feedback!"

    except Exception as e:
        return f"error occurred, the error is {e}."
```

update_knowledged_base

```
def update_knowledged_base() -> str:
    """\
    Useful for updating the knowledge base.

    Goal: Update the knowledge base.

    Output: Store the customer message to the knowledge base.
    """

    try:
        ...

        if customer_message is not in knowledge_base:
            store_customer_message_to_knowledge_base() # API call
            return "Store the customer message to the knowledge base
```

```
Successfully!"
        else:
            return "The customer message is already in the knowledge
base!"

        if update_knowledge_base() is True:
            return "knowledge_base_updated"
        else:
            return "knowledge_base_not_updated"

    except Exception as e:
        return f"error occurred, the error is {e}."
```

update_ticket

```
def update_ticket() -> str:
    """\
    Useful for updating the ticket if the customer is not satisfied with
the solution.

    Goal: Update the ticket.

    Output:
        - If the ticket is updated, return the "ticket_updated".
        - If the ticket is not updated, return the "ticket_not_updated".
    """

    try:
        ...
        get_history_log() # API call

        if update_ticket() is True: # API call
            return "ticket_updated"
        else:
            return "ticket_not_updated"

    except Exception as e:
        return f"error occurred, the error is {e}."
```

Scenario Example

1. My EC2 is broken, I cannot access my website.

    1. Time of rethink: 0
    2. Customer describe clear or not: clear
    3. Knowledge base: Yes
    4. Solution: Restart the EC2 instance
    5. Customer satisfy: Yes
    6. Open ticket: Yes

Steps:

1. `get_customer_message()`
2. `check_knowledged_base()`
3. `give_solution()`
4. `open_ticket()`
5. `process_customer_feedback()`
6. `get_customer_message()` to analyze the customer's feedback. if need to update the knowledge base, then `update_knowledged_base()`
7. Done

2. My EC2 is broken, I cannot access my website.

1. Time of rethink: 0
2. Customer describe clear or not: clear
3. Knowledge base: No
4. Solution: Not found
5. Customer satisfy: Yes
6. Open ticket: Yes

Steps:

1. `get_customer_message()`
2. `check_knowledged_base()`
3. `open_ticket()`
4. `process_customer_feedback()`
5. `get_customer_message()` to analyze the customer's feedback. if need to update the knowledge base, then `update_knowledged_base()`
6. Done

3. My EC2 is broken, I cannot access my website.

1. Time of rethink: 1
2. Customer describe clear or not: clear
3. Knowledge base: No
4. Solution: Not found
5. Customer satisfy: No
6. Open ticket: Yes

Steps:

1. `get_customer_message()`
2. `check_knowledged_base()`
3. `open_ticket()`
4. `get_customer_message()`
5. `rethink_solution()`
6. `open_ticket()`
7. `process_customer_feedback()`

8. `get_customer_message()` to analyze the customer's feedback. if need to update the knowledge base, then `update_knowledged_base()`
9. Done

3. My EC2 is broken, I cannot access my website.

1. Time of rethink: 0
2. Customer describe clear or not: not clear
3. Knowledge base: Yes
4. Solution: Restart the EC2 instance
5. Customer satisfy: Yes
6. Open ticket: Yes

Steps:

1. `get_customer_message()`
2. `get_customer_message()`
3. `check_knowledged_base()`
4. `open_ticket()`
5. `process_customer_feedback()`
6. `get_customer_message()` to analyze the customer's feedback. if need to update the knowledge base, then `update_knowledged_base()`
7. Done

4. My EC2 is broken, I cannot access my website.

1. Time of rethink: 4 (for example: the limit of rethink is 3)
2. Customer describe clear or not: clear
3. Knowledge base: Yes
4. Solution: Restart the EC2 instance
5. Customer satisfy: No
6. Open ticket: No

Steps:

1. `get_customer_message()`
2. `check_knowledged_base()`
3. `open_ticket()`
4. `get_customer_message()`
5. `rethink_solution()`
6. `open_ticket()`
7. `get_customer_message()`
8. `rethink_solution()`
9. `open_ticket()`
10. `escalate_to_TAM()` (Because the rethinking is more than 3 times)
11. `process_customer_feedback()`
12. `get_customer_message()` to analyze the customer's feedback. if need to update the knowledge base, then `update_knowledged_base()`

13. Done