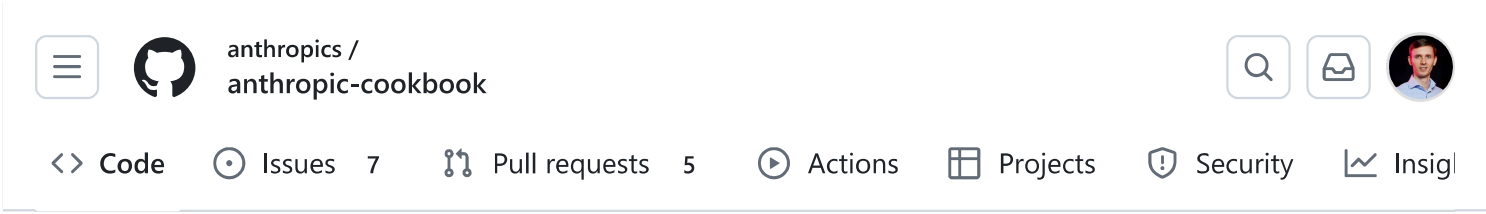

360 lines (360 loc) · 13.3 KB

Note-Saving Tool with Pydantic and Anthropic Tool Use

In this example, we'll create a tool that saves a note with the author and metadata, and use Pydantic to validate the model's response when calling the tool. We'll define the necessary Pydantic models, process the tool call, and ensure that the model's response conforms to the expected schema.

Step 1: Set up the environment

First, let's install the required libraries and set up the Anthropic API client.

In [ ]:
```
%pip install anthropic pydantic 'pydantic[email]'
```

In [2]:
```
from anthropic import Anthropic
from pydantic import BaseModel, EmailStr, Field
from typing import Optional

client = Anthropic()
MODEL_NAME = "claude-3-opus-20240229"
```

Step 2: Define the Pydantic models

We'll define Pydantic models to represent the expected schema for the note, author, and the model's response. This will allow us to validate and type-check the model's response when saving a note.

In [18]:
```
class Author(BaseModel):
    name: str
    email: EmailStr

class Note(BaseModel):
    note: str
    author: Author
    tags: Optional[list[str]] = None
    priority: int = Field(ge=1, le=5, default=3)
    is_public: bool = False

class SaveNoteResponse(BaseModel):
    success: bool
    message: str
```

Step 3: Define the client-side tool

Next, we'll define the client-side tool that our chatbot will use to save notes.

In [17]:
```
tools = [
    {
        "name": "save_note",
        "description": "A tool that saves a note with the author and metadata
        "input_schema": {
            "type": "object",
            "properties": {
                "note": {
                    "type": "string",
                    "description": "The content of the note to be saved."
                },
                "author": {
                    "type": "object",
                    "properties": {
                        "name": {
                            "type": "string",
                            "description": "The name of the author."
                        },
                        "email": {
                            "type": "string",
                            "format": "email",
                            "description": "The email address of the author."
                        }
                    },
                    "required": ["name", "email"]
                },
                "priority": {
                    "type": "integer",
                    "minimum": 1,
                    "maximum": 5,
                    "default": 3,
                    "description": "The priority level of the note (1-5)."
                },
                "is_public": {
                    "type": "boolean",
                    "default": False,
                    "description": "Indicates whether the note is publicly ac
                }
            },
            "required": ["note", "author"]
        }
    }
]
```

Step 4: Implement the note-saving tool

We'll create a dummy note saving function that just prints out that the note was saved successfully. If you actually want this note to be saved somewhere, you can implement this function.

In [23]:
```
def save_note(note: str, author: dict, priority: int = 3, is_public: bool = F
    print("Note saved successfully!")
```

In [24]:

```
def process_tool_call(tool_name, tool_input):
    if tool_name == "save_note":
        note = Note(
            note=tool_input["note"],
            author=Author(
                name=tool_input["author"]["name"],
                email=tool_input["author"]["email"]
            ),
            priority=tool_input.get("priority", 3),
            is_public=tool_input.get("is_public", False)
        )
        save_note(note.note, note.author.model_dump(), note.priority, note.is
        return SaveNoteResponse(success=True, message="Note saved successfull

def generate_response(save_note_response):
    return f"Response: {save_note_response.message}"
```

Step 6: Interact with the chatbot

Now, let's create a function to interact with the chatbot. We'll send a user message, process the tool call made by Claude, generate the response, validate the model's response using Pydantic, and return the final response to the user.

In [21]:

```
def chatbot_interaction(user_message):
    print(f"\n{'='*50}\nUser Message: {user_message}\n{'='*50}")

    messages = [
        {"role": "user", "content": user_message}
    ]

    message = client.messages.create(
        model=MODEL_NAME,
        max_tokens=4096,
        tools=tools,
        messages=messages
    )

    print(f"\nInitial Response:")
    print(f"Stop Reason: {message.stop_reason}")
    print(f"Content: {message.content}")

    if message.stop_reason == "tool_use":
```

```
        tool_use = next(block for block in message.content if block.type == "
        tool_name = tool_use.name
        tool_input = tool_use.input

        print(f"\nTool Used: {tool_name}")
        print(f"Tool Input: {tool_input}")

        save_note_response = process_tool_call(tool_name, tool_input)


        print(f"Tool Result: {save_note_response}")

        response = client.messages.create(
            model=MODEL_NAME,
            max_tokens=4096,
            messages=[
                {"role": "user", "content": user_message},
                {"role": "assistant", "content": message.content},
                {
                    "role": "user",
                    "content": [
                        {
                            "type": "tool_result",
                            "tool_use_id": tool_use.id,
                            "content": str(save_note_response),
                        }
                    ],
                },
            ],
            tools=tools,
        )
    else:
        response = message

    final_response = next(
        (block.text for block in response.content if hasattr(block, "text")),
        None,
    )
    print(response.content)
    print(f"\nFinal Response: {final_response}")

    return final_response
```

Step 7: Test the chatbot

Let's test our chatbot with a sample query to save a note.

In [26]:

```
chatbot_interaction("""
Can you save a private note with the following details?
Note: Remember to buy milk and eggs.
Author: John Doe (johndoe@gmail.com)
Priority: 4
""")
```

```
==================================================
User Message:
```

```
Can you save a private note with the following details?
Note: Remember to buy milk and eggs.
Author: John Doe (johndoe@gmail.com)
Priority: 4

==================================================

Initial Response:
Stop Reason: tool_use
Content: [ContentBlock(text='<thinking>\nThe relevant tool to use here is save_
note, as the request is to save a note with specific details.\n\nLet\'s go thro
ugh the parameters one-by-one:\n\nnote: The user provided the note content: "Re
member to buy milk and eggs."\nauthor: The user provided the author details: \n
{\n  "name": "John Doe",\n  "email": "johndoe@gmail.com"\n}\nis_public: While t
he user didn\'t explicitly specify, they asked for a "private note", so we can
infer is_public should be false.\npriority: The user specified a priority of
4.\n\nAll the required parameters have been provided or can be reasonably infer
red from the request. We have enough information to make the save_note call.\n
</thinking>', type='text'), ContentBlockToolUse(id='toolu_015iteV2eC1C7aUodbkot
fiS', input={'note': 'Remember to buy milk and eggs.', 'author': {'name': 'John
Doe', 'email': 'johndoe@gmail.com'}, 'is_public': False, 'priority': 4}, name
='save_note', type='tool_use')]

Tool Used: save_note
Tool Input: {'note': 'Remember to buy milk and eggs.', 'author': {'name': 'John
Doe', 'email': 'johndoe@gmail.com'}, 'is_public': False, 'priority': 4}
Note saved successfully!
Tool Result: success=True message='Note saved successfully!'
[ContentBlock(text='Your private note has been saved successfully with the foll
owing details:\n\nNote: Remember to buy milk and eggs. \nAuthor: John Doe (john
doe@gmail.com)\nPriority: 4\nVisibility: Private\n\nPlease let me know if you n
eed anything else!', type='text')]
```