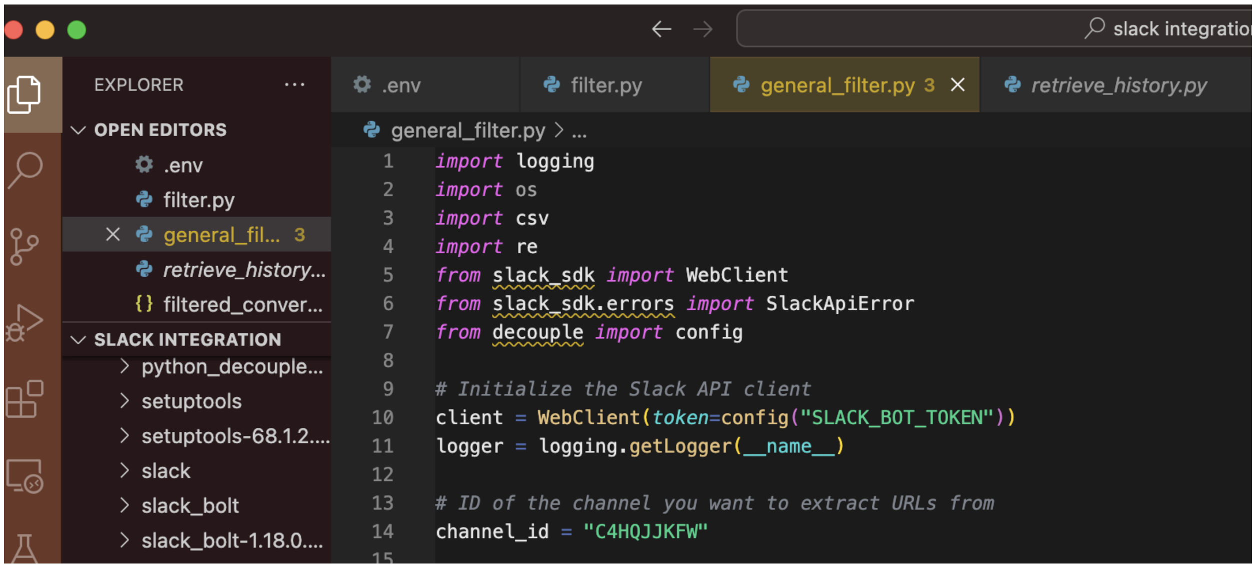
Project Description

My organization needs an integration app to extract the threat domains data from a Slack channel. I use Python SDK and Rest API of Slack to connect the apps and build an integration app

Integrate Slack and Python SDK

Slack allows admin to build their own apps or workflow to automate tasks. After configuring Slack, I got the `SLACK_BOT_TOKEN` to configure the SDK. In this screenshot, I tested on one of the organization's channels with `channel_id="C4HQJJKFW"` to run my code.

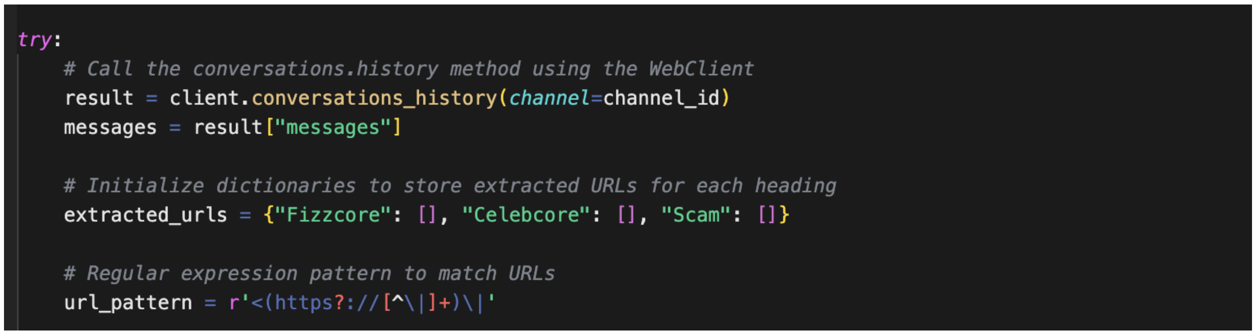


```python
import logging
import os
import csv
import re
from slack_sdk import WebClient
from slack_sdk.errors import SlackApiError
from decouple import config

# Initialize the Slack API client
client = WebClient(token=config("SLACK_BOT_TOKEN"))
logger = logging.getLogger(__name__)

# ID of the channel you want to extract URLs from
channel_id = "C4HQJJKFW"
```

Fetching Conversation History:



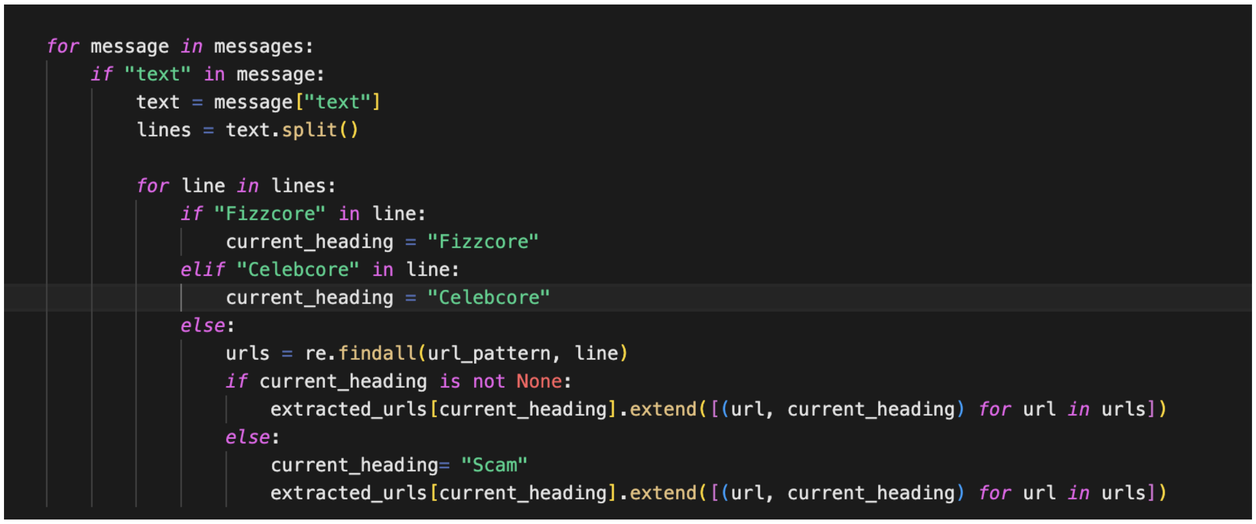
```python
try:
    # Call the conversations.history method using the WebClient
    result = client.conversations_history(channel=channel_id)
    messages = result["messages"]

    # Initialize dictionaries to store extracted URLs for each heading
    extracted_urls = {"Fizzcore": [], "Celebcore": [], "Scam": []}

    # Regular expression pattern to match URLs
    url_pattern = r'<(https?://[^\|]+)\|'
```

- I used the `conversations_history` method from a Slack `client` object to get the history of messages from a specific channel denoted by `channel_id`. It initializes an empty dictionary named `extracted_urls` with keys for three categories: "`Fizzcore`," "`Celebcore`," and "`Scam`." These keys are used to store the URLs found under each category.
- A regular expression pattern (`url_pattern`) is defined to match URLs in the text. This regex pattern looks for URLs enclosed within angle brackets (`< >`) followed by a pipe character (`|`).

Processing Messages:



```python
for message in messages:
    if "text" in message:
        text = message["text"]
        lines = text.split()

        for line in lines:
            if "Fizzcore" in line:
                current_heading = "Fizzcore"
            elif "Celebcore" in line:
                current_heading = "Celebcore"
            else:
                urls = re.findall(url_pattern, line)
                if current_heading is not None:
                    extracted_urls[current_heading].extend([(url, current_heading) for url in urls])
                else:
                    current_heading= "Scam"
                    extracted_urls[current_heading].extend([(url, current_heading) for url in urls])
```

- It iterates through each message in the retrieved conversation history. For each message, it checks if a "text" field is available in the message. It then splits the text into lines and processes each line.
- When it encounters a line containing "Fizzcore" or "Celebcore," it sets the `current_heading` variable to the respective category. For other lines that don't mention "Fizzcore" or "Celebcore," it uses a regular expression to find URLs within the text.

- If a heading has been set (`current_heading` is not `None`), it appends the found URLs with their respective category into the `extracted_urls` dictionary.
- If a line doesn't contain "Fizzcore" or "Celebcore" and no URLs are found, it assigns the current heading as "Scam" and appends any URLs found under this category.

Summary:

This project has various Python components to interact with the Slack API and extract URLs from specific channels. Using the `WebClient` object, it communicates with Slack, while the `Logger` object manages error and information logs. Employing a range of operators like assignment, membership, comparison, and iteration, the code navigates message content to categorize URLs into 'Fizzcore', 'Celebcore', and 'Scam' headings using lists and dictionaries. It heavily relies on diverse modules and libraries such as `logging`, `os`, `csv`, `re`, `slack_sdk`, and `decouple` for efficient functionality. Additionally, it employs control structures like conditional statements and exception handling to categorize URLs based on context and manage potential errors. Finally, the project utilizes CSV file operations to store the categorized URLs and their respective tags in a file named `filtered_urls.csv`. This comprehensive use of Python components demonstrates adeptness in API integration, data manipulation, error handling, and file operations within a practical context.