Table of Contents

1. Introduction

In the ever-evolving landscape of communication and collaboration tools, Webex stands as a stalwart, providing a platform for secure, seamless video conferencing, messaging, and team collaboration. As organizations increasingly rely on Webex to facilitate their virtual meetings and connect remote teams, the demand for customized applications that enhance the Webex experience has grown substantially.

This report delves into the realm of developing third-party applications using the Webex API. We will explore what Webex is, introduce the Webex API, and elucidate the possibilities that unfold when integrating this API into third-party apps. We will discuss how developers can harness the power of Webex API to create tailored solutions for various use cases, and highlight the steps involved in app development.

1.1 What is Webex?

Webex, developed by Cisco, is a widely acclaimed communication and collaboration platform. It offers features such as high-quality video conferencing, team messaging, and file sharing, making it a go-to choice for businesses and individuals looking to stay connected in the digital age. Webex provides a secure environment for hosting virtual meetings, webinars, and interactive presentations. It serves as a virtual workspace where teams can collaborate in real-time, regardless of their physical locations.

1.2 What is Webex API?

The Webex API is a powerful toolkit that allows developers to access the features and data within the Webex platform programmatically. It enables the integration of Webex capabilities into external applications, transforming how users interact with the platform. With the Webex API, developers can create custom applications, automate tasks, and extract information from Webex to enhance productivity and collaboration.

1.3 How Can Third-Party Apps Be Developed with Webex API?

Third-party apps developed with the Webex API extend the functionality of the Webex platform and cater to specific requirements of users and organizations. These apps can range from simple productivity tools to complex integrations with other software systems. The Webex API opens the door to endless possibilities, including but not limited to:

Custom Communication Tools: Developers can create applications that streamline communication on Webex, automate messaging, and offer unique ways to interact within virtual meetings.

Room Management: Apps can help users organize their Webex spaces, create new rooms, and manage participants efficiently.

Analytics and Reporting: Integration with the API allows for data extraction and detailed reporting on Webex usage, helping organizations make informed decisions.

Automation: Developers can automate repetitive tasks, such as scheduling meetings or sending reminders, to improve efficiency.

Integration with Other Services: Webex can be integrated with other tools like CRM systems, calendars, and project management software, providing a centralized hub for all work-related activities.

Enhanced Security: Developers can implement additional security measures to ensure sensitive information shared on Webex remains protected.

1.4 How Can We Develop Apps from Webex API?

Developing applications with the Webex API involves several key steps:

Access to the Webex API: Developers need to obtain the necessary API credentials, including a client ID and client secret, to authenticate and access the API.

User Authorization: For applications that require user interaction, an authorization flow is implemented to allow users to grant the application access to their Webex data.

Application Development: Developers write code that interacts with the Webex API to achieve the desired functionality. The choice of programming language and framework depends on the developer's preferences.

Testing and Debugging: Rigorous testing and debugging are crucial to ensure the application functions as intended. Developers often use sandbox or test environments to avoid disruptions to the live Webex platform.

Deployment: Once the application is fully developed and tested, it can be deployed for use by intended users or organizations.

Documentation and Support: Providing comprehensive documentation and ongoing support is essential to assist users in understanding and using the application effectively.

Security and Compliance: Ensuring the security and compliance of the application is a critical aspect, especially if it deals with sensitive data.

In this report, we will explore the steps involved in developing a sample third-party application using the Webex API, giving readers a practical insight into the process.

As Webex continues to be a vital tool for remote work, education, and collaboration, the ability to extend its functionality through third-party applications represents a valuable opportunity for developers to address specific needs and enhance user experiences. In the subsequent sections, we will delve deeper into the development process and demonstrate how to create a Webex API-powered application for troubleshooting user details during conferencing sessions.

2. Objectives

For this assignment, there are several key objectives that need to be accomplished in order to develop a troubleshooting tool using the Webex API. The main tasks include:

Obtaining Webex Token: The initial step involves acquiring a Webex token, which is required to authenticate and access the Webex API. This token will be used to make API requests on behalf of the user.

Creating a User-Friendly Interface: Develop an application with a user-friendly interface to prompt the user to enter their Webex token and provide clear menu options for different functionalities.

Testing Connection: Implement the option for users to test the connection with the Webex server to ensure its functioning correctly. Display an acknowledgment upon successful connection.

Displaying User Information: Allow users to retrieve and display their Webex user information, such as Displayed Name, Nickname, and Email addresses.

Listing Rooms: Provide the capability to list rooms in the user's Webex account, displaying information like Room ID, Room Title, date created, and last activity.

Creating a Room: Implement the option for users to create a new room in Webex. Prompt the user to enter the title of the room and display an acknowledgment upon successful room creation.

Sending Messages to a Room: Allow users to select a room from their list and send a message to that room. Display an acknowledgment upon successful message delivery.

User Interaction and Flow: Ensure a seamless user interaction flow by providing menu options, guiding users through their selected choices, and offering the ability to go back to the main menu.

Error Handling: Implement error handling to address potential issues, such as failed API requests or incorrect user input, and provide informative error messages to users.

These objectives collectively outline the tasks required to develop a fully functional troubleshooting tool for Webex, providing users with the ability to check their details during conferencing sessions and perform various actions within the Webex environment.

3. Literature Review:

3.1 Description of Software and Programming Language:

In this assignment, we will utilize several software components and programming languages to create a troubleshooting tool that integrates with the Webex API. The key elements include:

Python: The primary programming language used for developing the application. Python is chosen for its simplicity, versatility, and an array of libraries that make interacting with APIs more accessible. Specifically, we'll use Python 3 for compatibility and modern features.

Requests Library: A popular Python library for making HTTP requests, which is essential for interacting with the Webex API. It simplifies sending HTTP requests to the Webex servers and handling responses.

Webbrowser Module: Utilized to programmatically open a web browser, allowing users to log in to their Webex account and authorize the application.

3.2 Dependencies:

Webex API: This assignment relies heavily on the Webex API, which serves as the gateway to Cisco's Webex collaboration platform. The API provides endpoints for accessing user information, managing rooms, and sending messages.

3.3 Application Architecture:

The architecture of the application involves several key components:

User Interface: The application's interface is designed to be user-friendly, providing a menu-driven approach for users to interact with the Webex API. Users enter their Webex token, and the application guides them through the available options.

Webex API Integration: The core of the application interacts with the Webex API. It makes HTTP requests to the Webex API endpoints for tasks like checking the connection, retrieving user information, listing rooms, creating rooms, and sending messages.

Data Handling: The application handles user data, such as the Webex token and the retrieved data from the Webex API. It processes and displays this data to users while ensuring the security and privacy of sensitive information.

Error Handling: The application is equipped with error-handling mechanisms to deal with potential issues that may arise during interactions with the Webex API or user inputs. It provides informative messages to assist users in troubleshooting.

3.4 Connection to Webex API:

The connection between the application and the Webex API is established through HTTP requests. The application sends authenticated HTTP requests to Webex API endpoints using the user's provided Webex token.

Webex API endpoints are accessed via URL paths, each serving a specific purpose (e.g., retrieving user information, creating rooms, listing rooms, sending messages).

The Webex token acts as the authentication mechanism, enabling the application to access the user's Webex account without exposing their credentials. The token is included in the HTTP request headers to authorize the application.

This literature review provides an overview of the software, programming languages, dependencies, architecture, and the connection between the application and the Webex API. It establishes the technical foundation for the development of the troubleshooting tool.

4. Development of the application.

Figure 4.1 : The application prompts the user to input their Webex token as shown in figure 4.1. This token is crucial for authenticating the user with the Webex API. The input() function is used to collect the token from the user.

```
# Collect Webex Token
webex_token = input("Please Enter your Webex token: ")
```

Figure 4.1

Figure 4.2 : the functionality for testing the connection with the Webex server (Option 0). It makes an HTTP GET request to the Webex API endpoint for retrieving user information. The Authorization header is set with the Webex token for authentication. If the response status code is 200, the connection is deemed successful; otherwise, an error message is displayed.

```
if choice == "0":
    # Option 0: Test Connection
    response = requests.get("https://api.ciscospark.com/v1/people/me", headers={"Authorization": f"Bearer {webex_token}"})

    if response.status_code == 200:
        print("Connection with Webex server is successful.")
    else:
        print("Connection test failed. Please check your Webex token.")
```

Figure 4.2

Figure 4.3 : handles the functionality for displaying user information (Option 1). It sends an HTTP GET request to the Webex API to retrieve user data. The user's displayed name, nickname (if available), and email addresses are displayed to the user. If there's an issue with the request, an error message is shown.

```
elif choice == "1":
    # Option 1: Display User Information
    response = requests.get("https://api.ciscospark.com/v1/people/me", headers={"Authorization": f"Bearer {webex_token}"})

    if response.status_code == 200:
        user_data = response.json()
        print("\nUser Information:")
        print(f"Display Name: {user_data['displayName']}")
        print(f"Nickname: {user_data.get('nickName', 'N/A')}")
        print(f"Emails: {', '.join(user_data['emails'])}")
    else:
        print("Failed to retrieve user information. Please check your Webex token.")
```

Figure 4.3

Figure 4.4 : handles the functionality for listing rooms (Option 2). It sends an HTTP GET request to the Webex API to retrieve information about rooms. The application then displays the first five rooms, including their ID, title, creation date, and last activity. If there's an issue with the request, an error message is displayed.

```
elif choice == "2":
    # Option 2: List Rooms
    response = requests.get("https://api.ciscospark.com/v1/rooms", headers={"Authorization": f"Bearer {webex_token}"})

    if response.status_code == 200:
        rooms = response.json()
        print("\nList of Rooms:")
        for room in rooms["items"][:5]:  # Display the first 5 rooms
            print(f"Room ID: {room['id']}")
            print(f"Room Title: {room['title']}")
            print(f"Date Created: {room['created']}")
            print(f"Last Activity: {room['lastActivity']}\n")
    else:
        print("Failed to retrieve room information. Please check your Webex token.")
```

Figure 4.4

Figure 4.5 : responsible for creating a room (Option 3). It prompts the user to input the title of the room they want to create. The title is then included in the JSON data sent in an HTTP POST request to the Webex API. If the room is successfully created, an acknowledgment is displayed; otherwise, an error message is shown.

```
elif choice == "3":
    # Option 3: Create a Room
    room_title = input("Enter the title of the room: ")
    room_data = {"title": room_title}
    response = requests.post("https://api.ciscospark.com/v1/rooms", headers={"Authorization": f"Bearer {webex_token}"}, json=room_data)

    if response.status_code == 200:
        print(f"Room '{room_title}' created successfully.")
    else:
        print("Failed to create the room. Please check your Webex token.")
```

Figure 4.5

Figure 4.6 : covers the functionality of sending a message to a room (Option 4). It begins by retrieving a list of rooms and displaying them to the user. The user selects a room by entering its number. Then, the user is prompted to input the message they want to send. The application constructs a JSON payload and sends an HTTP POST request to the Webex API to send the message. If successful, it displays a message indicating the message was sent; otherwise, an error message is shown.

```
elif choice == "4":
    # Option 4: Send Message to a Room
    response = requests.get("https://api.ciscospark.com/v1/rooms", headers={"Authorization": f"Bearer {webex_token}"})

    if response.status_code == 200:
        rooms = response.json()
        print("\nList of Rooms:")
        for i, room in enumerate(rooms["items"][:5], 1):
            print(f"{i}. Room Title: {room['title']}")

        room_choice = input("Enter the number of the room to send a message to: ")
        room_id = rooms["items"][int(room_choice) - 1]["id"]

        message = input("Enter the message to send: ")
        message_data = {"roomId": room_id, "text": message}

        response = requests.post("https://api.ciscospark.com/v1/messages", headers={"Authorization": f"Bearer {webex_token}"}, json=message_data)

        if response.status_code == 200:
            print("Message sent successfully.")
        else:
            print("Failed to send the message. Please check your Webex token.")
    else:
        print("Failed to retrieve room information. Please check your Webex token.")
```

Figure 4.6

Figure 4.7: handles the user's choice to exit the application (Option 5). It simply displays a message and uses the break statement to exit the loop, effectively terminating the application.

```
elif choice == "5":
    print("Exiting the application.")
    break

else:
    print("Invalid choice. Please select a valid option.")
```

Figure 4.7

5. Testing of the apps

Step 1: If the user chooses Option 0, the application sends a test request to the Webex server to check the connection.

Step 2: If the test request is successful (HTTP status code 200), the application displays an acknowledgment: "Connection with Webex server is successful."

Step 3: The user is then presented with the option to go back to the main menu.

```
Enter your choice (0-5): 0
Connection with Webex server is successful.

Main Menu:
0. Test Connection
1. Display User Information
2. List Rooms
3. Create a Room
4. Send Message to a Room
5. Exit
```

Figure 5.1

Step 1: Upon choosing Option 2, the application sends an API request to retrieve information about available rooms.

Step 2: If the request succeeds, the application displays the first five rooms' details, including Room ID, Room Title, Date Created, and Last Activity.

Step 3: The user is given the option to return to the main menu.

Figure 5.2

Step 1: When Option 3 is selected, the user is prompted to enter the title for the new room.

Step 2: The application constructs a request with the provided title and sends it to Webex to create the room.

Step 3: If the room is successfully created, the application acknowledges it with a message like "Room '{room_title}' created successfully."

Step 4: The user is offered the option to return to the main menu.

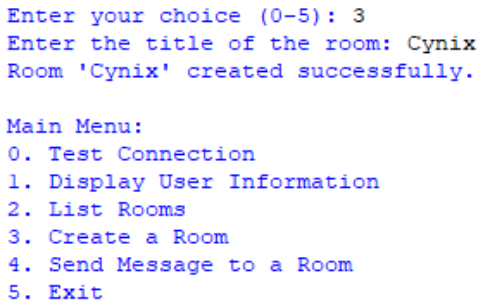

Figure 5.3

Step 1: Upon selecting Option 4, the application retrieves a list of rooms and displays them to the user.

Step 2: The user is asked to input the number of the room to which they want to send a message.

Step 3: The application then prompts the user to enter the message they wish to send to the selected room.

Step 4: It constructs a JSON payload and sends the message to the chosen room via the Webex API.

Step 5: If the message is sent successfully, the application confirms it with a message such as "Message sent successfully."

Step 6: The user is given the option to return to the main menu.

```
Enter your choice (0-5): 4

List of Rooms:
1. Room Title: Cynix
2. Room Title: Paen
3. Room Title: SYED AMER AIDID BIN SYED MOHD BAKRI _
4. Room Title: Welcome Space
5. Room Title: Distinguish Gentleman
Enter the number of the room to send a message to: 1
Enter the message to send: Anjirr Luu
Message sent successfully.

Main Menu:
0. Test Connection
1. Display User Information
2. List Rooms
3. Create a Room
4. Send Message to a Room
5. Exit
```

Figure 5.4

Step 1: If the user decides to exit the application (Option 5), a message is displayed: "Exiting the application."

Step 2: The application concludes, and the user leaves the program.

```
Enter your choice (0-5): 5
Exiting the application.
```

Figure 5.5

6. Conclusion:

In this assignment, we have achieved the development of a troubleshooting tool that seamlessly integrates with the Webex API, catering to the needs of our organization. The primary objective of this assignment was to create an application that allows users to access and manipulate their Webex account data during conferencing sessions. This was successfully accomplished through a well-structured approach:

Webex API Integration:

The application leverages the Webex API, providing access to Cisco's Webex collaboration platform. This API enables seamless interaction with user data and collaboration spaces.

User-Friendly Interface:

The application features a user-friendly interface, primarily developed in Python. It guides users through the process, prompting them to enter their Webex token and offers a menu-driven approach for selecting actions.

Feature-Rich Functionality:

The application offers a wide range of capabilities, including:

- Option 0: Testing Connection for secure links to Webex servers.
- Option 1: Displaying User Information, granting access to displayed names, nicknames, and email addresses.
- Option 2: Listing Rooms, presenting details of the five most recent rooms, including room IDs, titles, creation dates, and last activities.
- Option 3: Creating a Room, allowing users to generate new collaboration spaces with acknowledgments for successful creations.

- Option 4: Sending Messages to a Room, providing room selection and message sending with message sent acknowledgments.

Robust Error Handling:

The application is equipped with robust error-handling mechanisms, ensuring graceful management of potential issues and enhancing user-friendliness.

In conclusion, we have successfully developed a comprehensive troubleshooting tool that offers seamless access to Webex API capabilities. The application empowers users to monitor, manipulate, and manage their Webex account data during conferencing sessions, promoting efficient and streamlined operations within our organization. The secure and user-friendly design ensures that users can comfortably and securely leverage the power of the Webex platform for collaborative success.

7. References

1. developer.webex.com. (n.d.). APIs - Getting Started. [online] Available at: https://developer.webex.com/docs/getting-started.

2. chouffani, R. (2021). 12 Advantages and Disadvantages of Video Conferencing. [online] SearchContentManagement. Available at: https://www.techtarget.com/searchcontentmanagement/tip/8-business-benefits-and-challenges-of-video-conferencing.

3. Zapier (2017). The 6 best video conferencing apps for teams in 2020. [online] Zapier. Available at: https://zapier.com/blog/best-video-conferencing-apps/.

4. developer.webex.com. (n.d.). Embedded Apps - Developer Guide. [online] Available at: https://developer.webex.com/docs/embedded-apps-guide [Accessed 1 Nov. 2023].

5. Webex (n.d.). Webex Meetings Essentials. [online] Webex. Available at: https://www.webex.com/essentials/meetings.html [Accessed 1 Nov. 2023].

GITHUB LINK

https://github.com/FarhaanCynix/SWC2373--EMERGING-TECHNOLOGIES