



Università degli Studi di Messina

integration between the scraper and a Telegram bot

prof: Lorenzo Carnevale

student: Mujibullah Rabin

content:

- **Introduction**
- **Create Telegram bot.**
- **Telegram bot token**
- **Create Scopus API key**
- **Explanation of the code**
- **Running the code**
- **Result in Telegram bot**
- **Output in the Telegram**

Introduction:

This project enables you to access and utilize information from the Scopus database, a valuable resource in the field of educational research. By using a Scopus API key, we've created a tool that connects to the database and communicates the information through a Telegram bot using a unique token.

This project offers different ways to find information:

1. **Author Search:** This project has the ability to display a list of authors with similar names when you enter an author's name. You can then choose the specific author you're interested in and view their details, such as their workplace, published papers, expertise areas, Scopus ID, and more.
2. **Scopus ID Search:** Instead of using names, you can search for authors by entering their unique Scopus ID. This method provides comprehensive information about the author associated with that ID.
3. **Document Retrieval:** If you're looking for a particular document, you can simply input its Scopus ID, and the project will provide you with information about the authors, document title, and an introduction to the document.

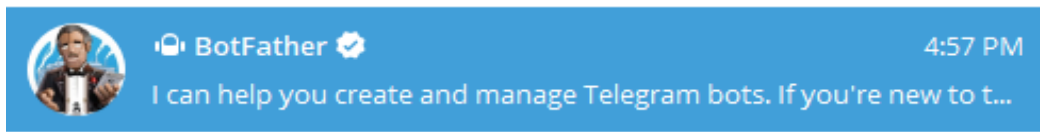
This project offers a variety of options to make it easy for you to access the specific information you need, whether you're interested in authors, documents, or their details. In the following sections of this report, we will explain how this project operates and how it can be beneficial for educational research.

First of all, we need to create a Telegram bot. To create one, follow these steps:

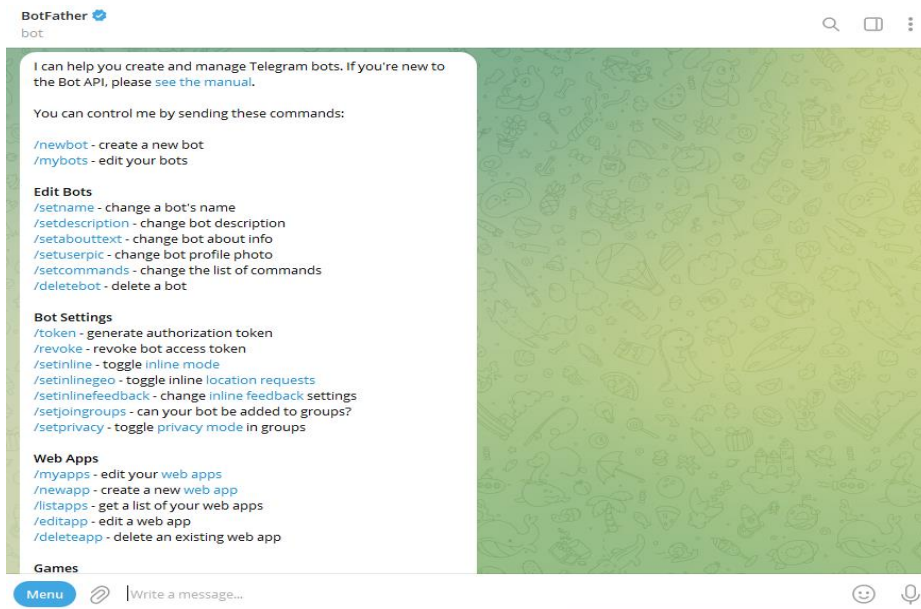
1. Open the Telegram application on your device.



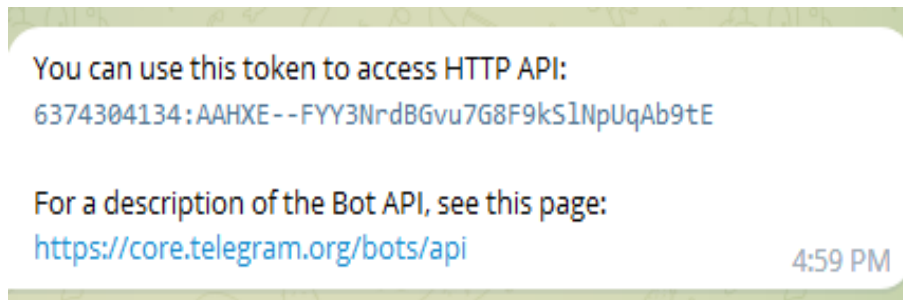
2. In the Telegram application, search for a bot called "BotFather." This bot is provided by Telegram and allows you to create and manage other bots.



3. Start a chat with BotFather by sending a message to it.
4. Follow the instructions provided by BotFather to create your bot. You can typically use the **/newbot** command to initiate the bot creation process. BotFather will ask you to choose a name for your bot and provide you with a unique username (ending in "bot") for your bot.



- 5.
6. Once you've completed these steps, your Telegram bot will be created, and you'll receive a token. This token is crucial for interacting with the Telegram Bot API and making your bot functional.



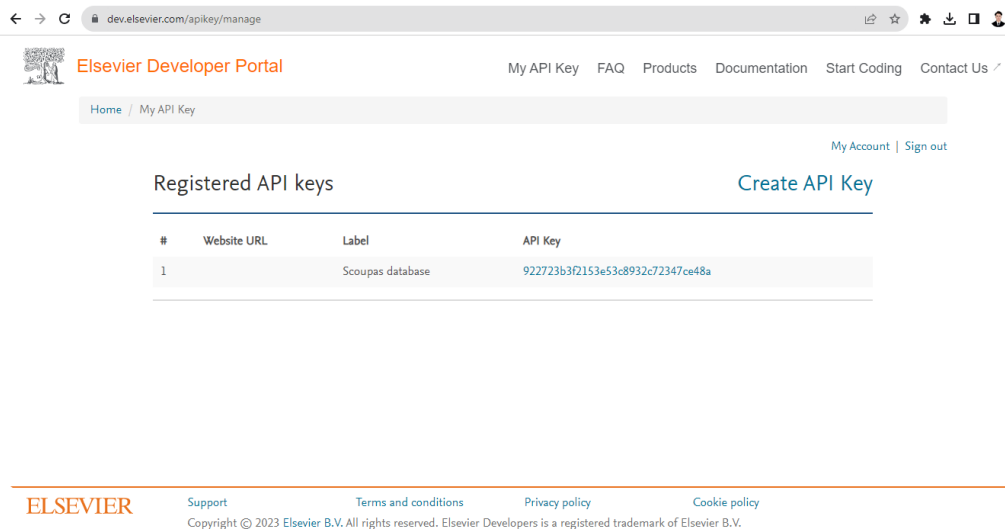
By following these steps, you can create and set up a Telegram bot using BotFather within the Telegram application.

Create a Scopus API key:

- When you make a Telegram bot and get a special token for it, the next thing to do is create something called an "API key." An API key is like a unique passcode that allows different computer programs to talk to each other and share information.
- In the case of the Scopus Scraper application, you need this API key. It's like a special ID card for your application. This ID card helps the Scopus database know that your application is allowed to use it, and it ensures that only the right applications or people can access the Scopus database.

To get the API key we have to signup

<https://dev.elsevier.com/> in this link and get our API key.



The screenshot shows the 'dev.elsevier.com/apikey/manage' page. The header includes the Elsevier logo and 'Elsevier Developer Portal'. Navigation links include 'My API Key', 'FAQ', 'Products', 'Documentation', 'Start Coding', and 'Contact Us'. A breadcrumb trail shows 'Home / My API Key'. On the right, there are links for 'My Account' and 'Sign out'. The main content area is titled 'Registered API keys' and features a 'Create API Key' link. Below this is a table with one entry:

#	Website URL	Label	API Key
1		Scoupas database	922723b3f2153e53c8932c72347ce48a

The footer contains the Elsevier logo, links for 'Support', 'Terms and conditions', 'Privacy policy', and 'Cookie policy', and a copyright notice: 'Copyright © 2023 Elsevier B.V. All rights reserved. Elsevier Developers is a registered trademark of Elsevier B.V.'

Codes:

To start writing code, we need to add some extra tools to our coding environment, whether we're using Python or another programming language. These extra tools are called "extensions," and they help us do specific tasks.

1. **Requests:** This is like a tool that allows our code to talk to other computers on the internet. We use it when we want our program to ask a website for some information, like getting weather data.
2. **JSON:** Imagine JSON as a way to organize information in a special format that both computers and people can easily understand. It's like putting data into neat boxes so our program can work with it smoothly.

```
1 import requests
2 import json
3 from telegram import Update, ReplyKeyboa
4 from telegram.ext import Updater, Command
```

Now, these two extensions are used for connecting our code to Telegram and making changes to the Telegram bot. They help us interact with Telegram through our Python code:

1. **Telegram Bot:** This extension allows us to connect our Python code with the Telegram app. It's like a bridge that lets our code talk to Telegram.
2. **telegram. Ext:** With this extension, we can create special features like conversation handlers and more for our Telegram bot. It helps us manage how our bot responds to different messages and commands.

By adding these extensions, we can make our Python program work together with Telegram and customize our bot's behavior.

Now, we will begin by providing our Scopus API key. This key grants us permission to access the Scopus database. Additionally, we need to define the states for conversation handling.

Next, we should create a function to extract author information from Scopus using their ID. It's crucial to ensure the correct URL key is specified so that the API key can successfully access the URL and retrieve the desired information.

```
5
6 SCOPUS_API_KEY = "922723b3f2153e53c8932c72347ce48a"
7
8 SEARCH_BY_NAME, SEARCH_BY_ID, SEARCH_BY_NAME_ID, SEARCH_BY_DOCUMENT = range(4) # D
9
10 def extract_author_info(author_id):
11     url = f"https://api.elsevier.com/content/search/author?query=AU-ID({author_id})"
12     headers = {
13         "Accept": "application/json",
14         "X-ELS-APIKey": SCOPUS_API_KEY
15     }
16
```

After providing all the necessary details, we can explore another option, which is searching for an author by their name. To achieve this, we will split the full name into first name and last name, allowing us to search for both simultaneously in the same query. Then, we'll provide the appropriate URL and Scopus ID, enabling access to the database and retrieval of the desired information.

```
25
26 def extract_author_info_by_name(author_name):
27     # Split the input into first and last names
28     parts = author_name.split()
29     if len(parts) != 2:
30         return []
31
32     first_name, last_name = parts[0], parts[1]
33
34     url = f"https://api.elsevier.com/content/search/author?query=AUTHFIRST({first_name"
35     headers = {
36         "Accept": "application/json",
37         "X-ELS-APIKey": SCOPUS_API_KEY
38     }
39
40     try:
41         response = requests.get(url, headers=headers)
42         response.raise_for_status()
43         data = json.loads(response.text)
44         authors = data["search-results"]["entry"]
45         return authors
46     except requests.exceptions.RequestException as e:
47         return []
```


Now that we've completed our first option and obtained the initial set of data, it's time to move on to the second option: searching by document ID. In this phase, we'll follow the same process by using our API key to access the relevant URL and retrieve information directly from the database.

```
49 def extract_document_info(document_id):
50     url = f"https://api.elsevier.com/content/abstract/scopus_id/{document_id}"
51     headers = {
52         "Accept": "application/json",
53         "X-ELS-APIKey": SCOPUS_API_KEY
54     }
55
56     try:
57         response = requests.get(url, headers=headers)
58         response.raise_for_status()
59         data = json.loads(response.text)
60         doc_info = data["abstracts-retrieval-response"]["coredata"]
61         return doc_info
62     except requests.exceptions.RequestException as e:
63         return None
64
```

So in this code as we ask for specific information we want to extract as you can see in the code document info including the abstracts (introduction of the document).

So we go step by step

To extract each piece of information, we need to grant access via the URL and code it accordingly to retrieve the desired data. For example, to extract metrics information, we'll write the code in a way that enables access to the database and retrieves the metrics data via the provided URL.

```
64
65 # Define a function to extract author metrics from Scopus
66 def extract_author_metrics(author_id):
67     url = f"https://api.elsevier.com/content/author/metrics?author_id={author_id}"
68     headers = {
69         "Accept": "application/json",
70         "X-ELS-APIKey": SCOPUS_API_KEY
71     }
72
73     try:
74         response = requests.get(url, headers=headers)
75         response.raise_for_status()
76         data = json.loads(response.text)
77         metrics = data.get("author-retrieval-response", {}).get("coredata", {})
78         return metrics
79     except requests.exceptions.RequestException as e:
80         return {}
81
```

And here we go for chat with bot with these codes' handler. This code is part of a chatbot program. It defines a function that responds to a user's choice among different search options. When the user makes a choice, such as "Search by Name" or "Search by ID," the bot replies with instructions related to that choice. The function also returns a value indicating the chosen search type, which is likely used to guide the bot's subsequent actions in handling the search request. And in summary this code plays a crucial role in a

chatbot's interaction flow. It captures and responds to the user's choice of search type, guiding the conversation in the direction of the chosen search method while also providing clear instructions to the user.

```
88 # Function to handle the user's choice of search
89 def search_choice(update: Update, context: CallbackContext):
90     user_choice = update.message.text
91
92     if user_choice == "Search by Name":
93         update.message.reply_text("Please enter the author's first name and last name")
94         return SEARCH_BY_NAME
95     elif user_choice == "Search by ID":
96         update.message.reply_text("Please enter the author's Scopus ID:")
97         return SEARCH_BY_ID
98     elif user_choice == "Search by Document":
99         update.message.reply_text("Please enter the document ID:")
100         return SEARCH_BY_DOCUMENT
101
```

And here is another part of the code search by name this part of code relates to telegram that control the message and choosing the author from similar authors name

This function is responsible for handling user input when they want to search for an author by name.

1. **User Input:** The function receives an **update** object, which contains the user's message. In this case, the user is expected to provide the name of an author they want to search for.
2. **Search Author Info:** It calls a function called **extract_author_info_by_name** to search for author information based on the name provided by the user.
3. **Response Handling:**

- If authors are found with the given name, it prepares a list of author names and asks the user to choose one.
- It stores the list of authors in the user's context data.
- It returns **SEARCH_BY_NAME_ID** (which seems to be a constant) to indicate that the next step is to choose an author from the list.
- If no authors are found or an error occurs, it notifies the user and ends the conversation.

Code Block 2 - search_choice Function:

This function handles the user's choice when they are asked to select an author from a list.

1. **User Choice:** It takes the user's choice as input, which should be a number corresponding to the author they want to select.
2. **Processing the Choice:**
 - It checks if the user's choice is a valid number.
 - If it's valid, it retrieves the list of authors from the user's context.
 - It uses the chosen number to select a specific author from the list.
 - It then extracts additional information about the selected author, such as their documents and metrics.

These functions appear to be part of a conversational system or chatbot where users can search for authors and retrieve information about them. The first function initiates the search based on the author's name, while the second function handles the user's choice of a specific author from the list of search results.

```

101
102 ~ def search_by_name(update: Update, context: CallbackContext):
103     author_name = update.message.text
104     authors = extract_author_info_by_name(author_name)
105
106 ~     if authors:
107         # Prepare a list of author names for the user to choose from
108         author_names = [f"{author['preferred-name']['given-name']} {author['preferred-
109 ~         update.message.reply_text("Multiple authors found with the same name. Please c
110         "\n".join([f"{index + 1}. {name}" for index, name in
111
112         # Set the list of authors and the state in the user's context
113         context.user_data['authors'] = authors
114         return SEARCH_BY_NAME_ID
115 ~     else:
116         update.message.reply_text("Author information not found or an error occurred.
117
118         return ConversationHandler.END
119
120 ~ def search_choice(update: Update, context: CallbackContext):
121     user_choice = update.message.text
122

```

And in the last part of the code, we will start the bot and using some functionalities to have conversation with the bot.

in this part we write our telegram bot code. And start bot from this part.

This code creates a chatbot that can interact with users on the Telegram messaging platform. Here's what it does:

1. **Bot Setup:** It prepares the bot for action. You need to replace a certain part of the code with your bot's unique token to identify it on Telegram.
2. **Conversation Handler:** It sets up a system for managing conversations with users. The bot understands different states like "SEARCH_BY_NAME," "SEARCH_BY_ID," etc. Each state corresponds to a different part of the conversation.
3. **Handling Messages:** Depending on the state of the conversation, the bot knows which functions should respond when a user sends a

message. For example, if the user is searching by name, the bot will call a function to handle that specific search.

4. **Adding Handlers:** It connects these conversation handlers to the bot so that it knows how to respond to user messages.
5. **Starting the Bot:** The bot is started, and it keeps running, waiting for messages from users. When a user sends a message, the bot processes it according to the conversation state and the defined handlers.

In summary, this code is a framework for creating a chatbot that can have conversations with users on Telegram. It listens to user messages, responds appropriately based on the conversation context, and can guide users through different parts of the conversation flow.

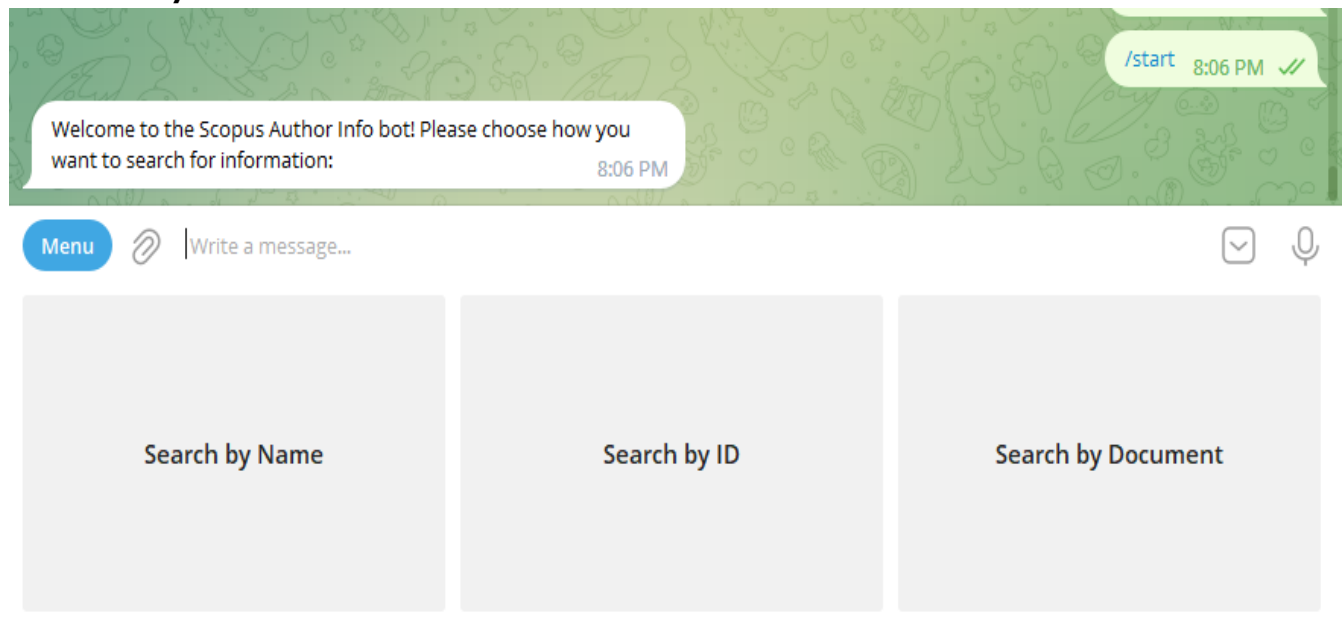
```
263 def main():
264     updater = Updater(token="6374304134:AAHXE--FYY3NrdBGvu7G8F9kS1NpUqAb9tE", use_cont
265     dispatcher = updater.dispatcher
266
267     # Define conversation handler
268     conv_handler = ConversationHandler(
269         entry_points=[CommandHandler('start', start)],
270         states={
271             SEARCH_BY_NAME_ID: [MessageHandler(Filters.text & ~Filters.command, search
272             SEARCH_BY_NAME: [MessageHandler(Filters.text & ~Filters.command, search_by
273             SEARCH_BY_ID: [MessageHandler(Filters.text & ~Filters.command, search_by_i
274             SEARCH_BY_DOCUMENT: [MessageHandler(Filters.text & ~Filters.command, searc
275         },
276         fallbacks=[],
277     )
278
279     # Add handlers to the dispatcher
280     dispatcher.add_handler(conv_handler)
281
282     # Start the bot
283     updater.start_polling()
284     updater.idle()
```

Running the code:

When we execute the code, all of its commands will be executed, and the results will be sent to a Telegram bot. This means that whatever we write in the code will appear in the Telegram bot as output.

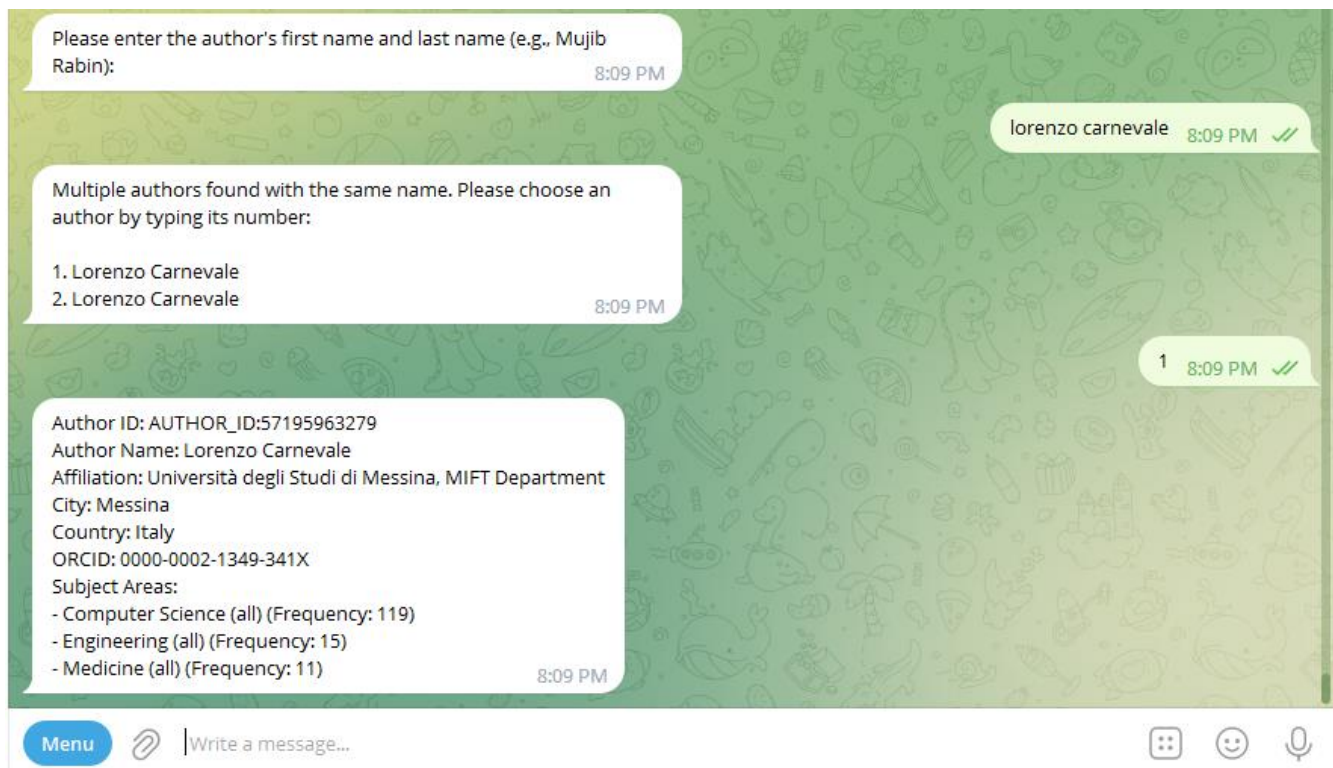
Result in telegram bot:

After all, when we run the code, we can open our Telegram bot, and we'll notice that the functions we've implemented are operational. Initially, after issuing the 'start' command, we will be presented with three options: search by name, search by ID, and search by document ID.



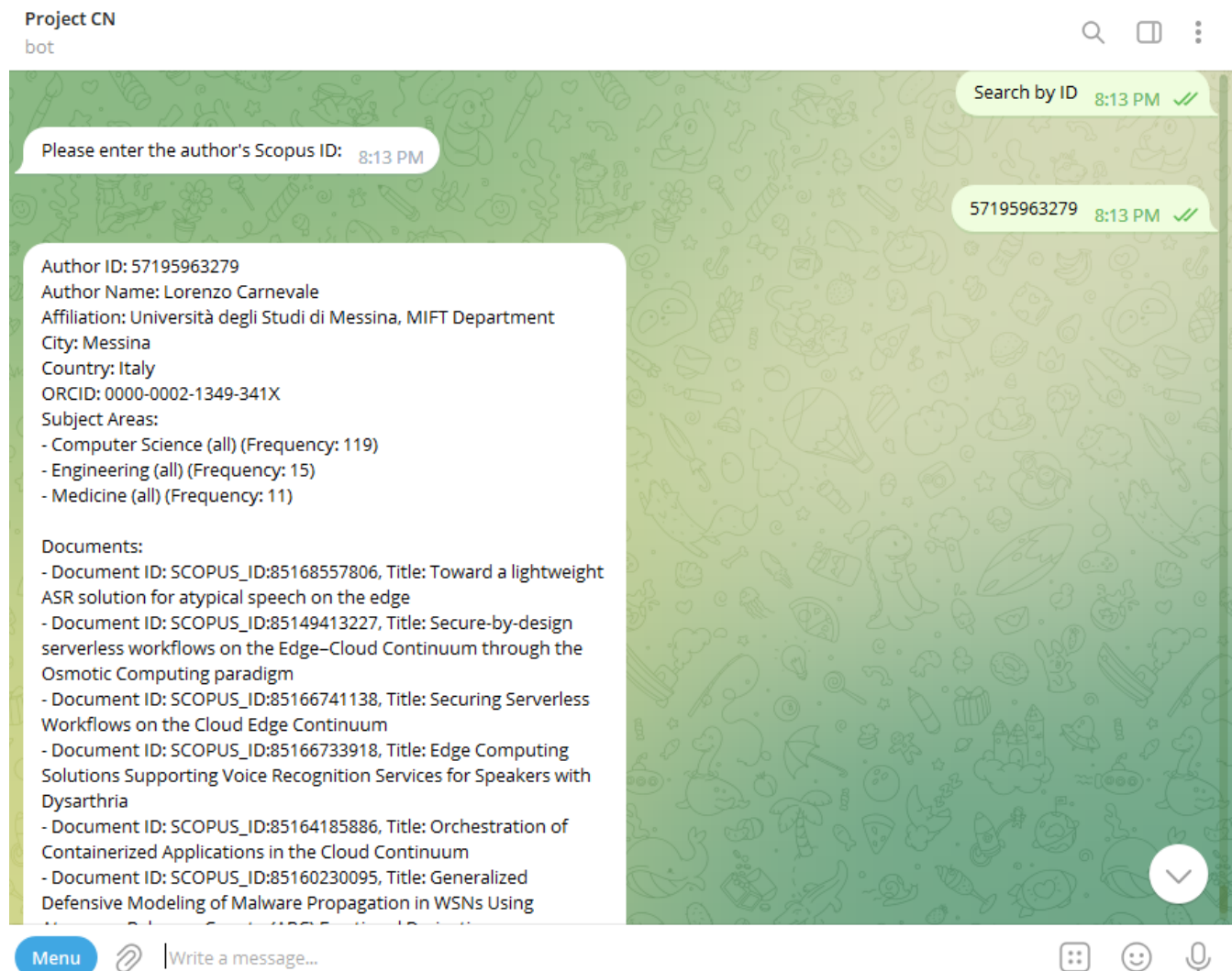
By choosing any of them we will be able to get information.

Furthermore, when we select the 'search by name' option, we will be prompted to enter a name. After entering the name, we will receive a list of all authors with the same name. Upon choosing one of them, we will have access to their information.



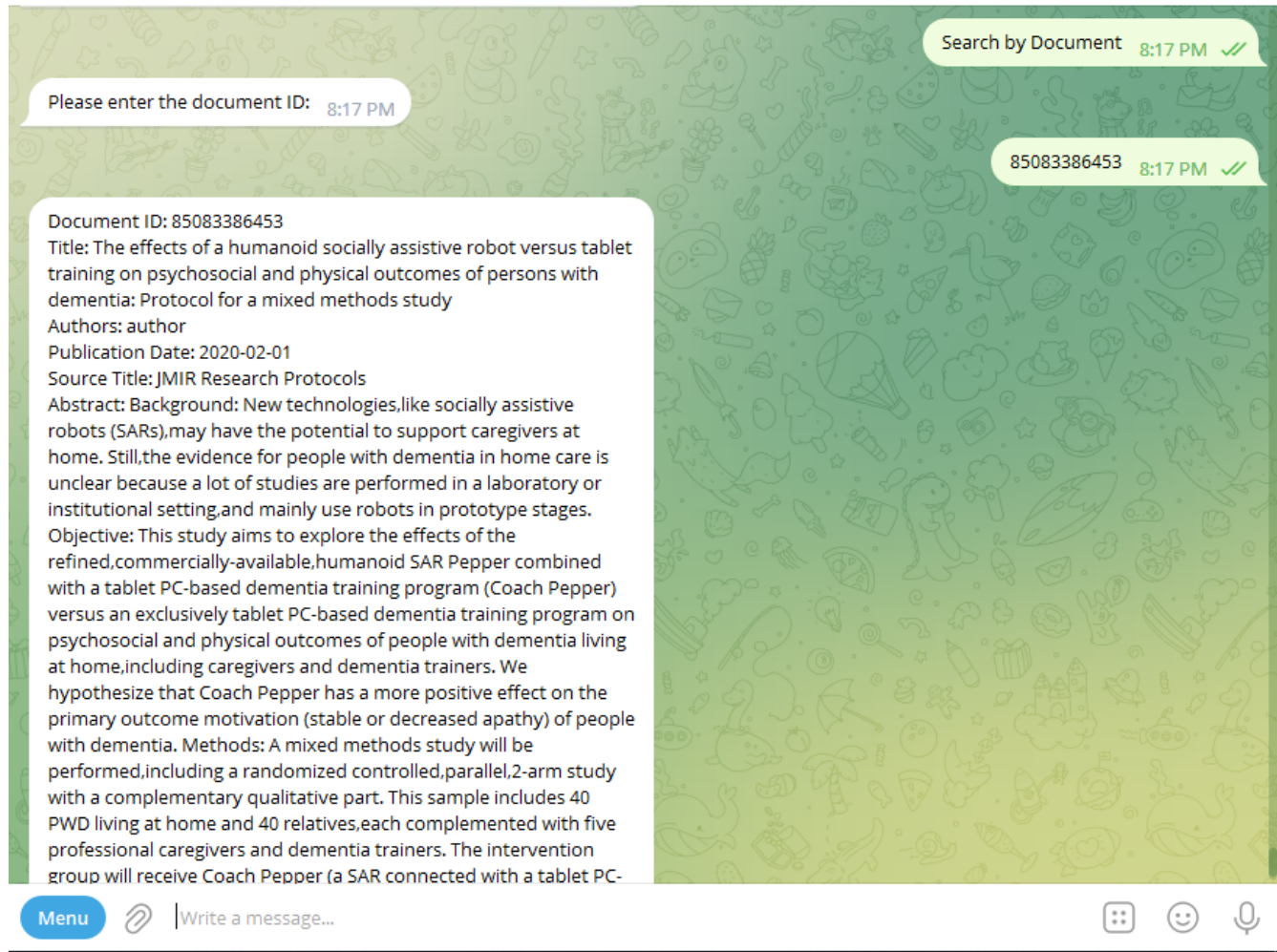
As you can see here, I entered the author's name and there were two authors with the same name, and I was able to choose one by choosing a corresponding number. And I got all the information by name.

Now, let's explore the second option: 'Search by Scopus ID of the author.' When we input the Scopus ID, we will receive comprehensive information, including the author's documents and their respective document IDs. This document ID can then be used in another option to retrieve detailed information about the documents.



So here we can see that after entering Scopus id of the author we are able to see all the information including the each document of the author.

Finally, for the last option, 'Searching for Documents,' we have the capability to input any document's Scopus ID to retrieve its information, as specified in our implemented code.



And here we searched the document Scopus id, and we got all the information, as document id, title, author, year of publication, source title, and abstract of the document.

Sites and links related to the project:

- <https://dev.elsevier.com/>
- <https://www.scopus.com/search/form.uri?display=basic#basic>
- https://t.me/CN_pro_bot bot link you can enter the bot and use it.