

# Multithreaded News Client/Server Information System

---

## Project Description

The Multithreaded News Client/Server is a simple Information System that aims to enable a secure exchange of information about recent news between clients and the server. The server retrieves the news from <https://newsapi.org/> depending on the client's request, where it can manage the connection with multiple clients in the same time. However, the system provides the user with a menu so it can choose between headlines and sources easily and provide a detailed response if required.

## Semester

S1 2024-2025

## Group

**Group name:** B4 **Course Code:** ITNE352

**Section:** 02

**Students:** Israa Isa Ahmed Altaitoon (ID: 202206492)

Zainab Hasan Isa Alobed (ID: 202206986)

## Table of Contents

1. [Requirements](#)
2. [How to](#)
3. [The Scripts](#)
4. [Additional Concepts](#)
5. [Acknowledgments](#)
6. [Conclusion](#)
7. [Resources](#)

## Requirements

Follow these steps to set up the project locally:

**Any disruption in the internet connection will prevent the system from functioning properly**  
**Ensure you are connected to the internet so API can work successfully**

1. Clone the repository:

```
git clone https://github.com/Zainab-Alobed/ITNE352-Project-Group-B4
```

2. Install required libraries:

```
pip install -r required.txt
```

### 3. Run the server.py

```
python server.py
```

### 4. Run the client.py

```
python client.py
```

## How to run the system:

### Run the server:

#### 1. Navigate to the server directory:

```
cd server
```

#### 2. Start the server:

```
python server.py
```

### Run the client:

#### 1. Navigate to the client directory:

```
cd client
```

#### 2. Start the client:

```
python client.py
```

### Interacting with the server:

1. The user will be asked about his name, and send it to the server
2. The main menu will be displayed in client side that contains three options (the user must input a valid number 1-3):

1. Headlines
  2. Sources
  3. Quit
3. A submenu of either headlines or sources will be displayed depending on the user choice
4. Later on, a maximum of 15 article will be displayed to provide the user the ability to request detailed information or go back to the main menu (Each request will be directly send to the server and print the response back in the client side)
5. The user can select (3) Quit to terminate the program

## The scripts

### Client script

- Purpose:  
interaction with server (Sends the user requests to the server and displays response)
- Functions:
  1. Create a TCP socket using IPv4 to connect to the server

```
# Create a TCP socket using IPv4
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_s:
    with context.wrap_socket(client_s, server_hostname="myserver") as cs:
        try:
            cs.connect((get_local_ip(), 5353))

            if isinstance(cs, socket.socket):
                connection(cs)
```

2. `get_user_name()`: ask user about his name and send it to the server

```
def get_user_name(cs):
    print("Hi!")
    while True:
        user_name = input("\nEnter your name (Only letters are allowed): ").strip()
        if re.match("^[a-zA-Z]+(?:[a-zA-Z]+)*$", user_name):
            break

        print("Invalid name. Only letters are allowed. Please try again.")

    cs.sendall(user_name.encode("utf-8"))
    print(f"Welcome {user_name}!\n")
```

3. `main_menu()`: display the main menu

```
def main_menu():
    main_menu = {"1": "headlines", "2": "sources", "3": "Quit"}
    while True:
        print("\nMain menu:")
        for key, value in main_menu.items():
            print(f"{key}. {value}")

        main_selection = input("Select an option: ").strip()

        if main_selection in main_menu:
            main_desc = main_menu[main_selection]
            break
        else:
            print("Invalid selected number! Try again.")

    return main_selection, main_desc
```

4. Headlines\_menu() + Sources\_menu(): display either headlines menu or sources menu depending on the user choice.

```
def Headlines_menu():
    Headlines = {
        "1": "keywords",
        "2": "category",
        "3": "country",
        "4": "all",
        "5": "main",
    }
    print("\n---- Headlines menu ----")
    for id, option in Headlines.items():
        print(f"{id} - {option}")

    Headlines_selection = input("\nSelect your option: ").strip()

    if Headlines_selection in Headlines:
        Headlines_desc = Headlines[Headlines_selection]
        return Headlines_selection, Headlines_desc
    else:
        print("Invalid selected number! Returning to main menu...")
        return -1, -1

"""
```

```
def Sources_menu():
    sources = {
        "1": "category",
        "2": "country",
        "3": "language",
        "4": "all",
        "5": "main",
    }

    print("\n---- Sources menu ----")
    for id, option in sources.items():
        print(f"{id} - {option}")

    sources_selection = input("Select your option: ").strip()

    if sources_selection in sources:
        source_desc = sources[sources_selection]
        return sources_selection, source_desc
    else:
        print("Invalid selected number! Returning to main menu...")
        return -1, -1
```

5. response(cs, main\_selection): display response of the user request

```
5 def response(cs, main_selection):
6     try:
7         # Check if the response is JSON which means a valid category, or country, or language
8         response = receive_complete_data(cs)
9         dict = json.loads(response)
10        n = len(dict) + 1 # Allow 'back' to the main menu option to the user
11        counter = 1
12
13        for article in dict:
14            if main_selection == "1": # Headlines menu
15                print(f"No. ({counter})")
16                print(f"Source name: {article['name']} ")
17                print(f"author: {article['author']} ")
18                print(f"title: {article['title']} \n")
19            elif main_selection == "2": # sources menu
20                print(f"No. ({counter})")
21                print(f"Source name: {article['name']} \n")
22            counter += 1
23        print(f"{n}. Back to the main menu\n")
24
25        return n
26
27    # Otherwise: it is not a JSON response which means invalid category, or country, or language.
28    # Print an appropriate message.
29    except json.JSONDecodeError:
30        print(f"{response}")
31        return -1
32
```

6. detailed\_response(cs, main\_selection): display of user detailed request if required

```

"""
def detailed_response(cs, main_selection):
    try:
        detailed_response = receive_complete_data(cs)
        detailed_response_dict = json.loads(detailed_response)
        source_name = detailed_response_dict.get("source", {}).get(
            "name", "Unknown Source"
        )

        url = detailed_response_dict.get("url", "No URL")
        description = detailed_response_dict.get("description", "No Description")
        print("Source name: ", source_name)
        print("URL: ", url)
        print("Description: ", description)

        if main_selection == "1":
            author = detailed_response_dict.get("author", "Unknown Author")
            title = detailed_response_dict.get("title", "No Title")
            published_at = detailed_response_dict.get("publishedAt", "Unknown Date")

            # Handle date and time extraction
            if "T" in published_at:
                date, time = published_at.split("T")
                print("Publish date: ", date)
                print("Publish time: ", time)
            else:
                print("Published at: ", published_at)
            print("Author: ", author)
            print("Title: ", title)

        else:
            country = detailed_response_dict.get("country", "Unknown Country")
            category = detailed_response_dict.get("category", "Unknown Category")
            language = detailed_response_dict.get("language", "Unknown Language")
            print("Country: ", country)
            print("Category: ", category)
            print("Language: ", language)

    except json.JSONDecodeError:
        print(detailed_response)
        return -1

```

## Server script

- Purpose:  
The server receives requests from the client for headlines/sources from newsapi. The server then will retrieve the requested data from an appropriate API endpoint. After getting the response, the server will save the response into a file named client name and the requested option then prepare a list containing brief information about a maximum of 15 headlines/sources and send it to the client.  
The client can choose a specific headline/source from the list to get more information about it, which will also be prepared by the server. The server will keep getting requests from a maximum of 3 clients.
- packages: re, socket, json, threading, requests, ssl, os
- Functions:
  1. get\_headlines + get\_all\_headlines  
Those two functions are responsible for retrieving headlines from API endpoints /everything and //top-headlines

```
def get_headlines(option, value):
    """
    This function is for getting headlined from the endpoint
    /top-headlines with specific criteria chosed by the client
    with the using of params
    """
    # You, 6 hours ago * improve structure of server and commenting

    params = {"apiKey": API_KEY, "pageSize": 15}
    if option == "keywords":
        params["q"] = value
    elif option == "category":
        params["category"] = value
    elif option == "country":
        params["country"] = value

    try:
        response = requests.get(f"{BASE_URL}/top-headlines", params=params)
        if response.status_code != 200:
            """
            if code is not 200, then the request was not accepted
            """
            print(f"Error with API: {response.status_code}, {response.text}")
            return {"error": "API error"}
        return response.json()

    except Exception:
        return {"API_error": "API error, check the connection"}

def get_all_headlines():
    """
    All headlines have different endpoint which is /everything
    """
    params = {"apiKey": API_KEY, "pageSize": 15, "language": "en", "q": "news"}
    try:
        response = requests.get(f"{BASE_URL}/everything", params=params)
        if response.status_code != 200:
            print(f"Error with API: {response.status_code}, {response.text}")
            return {"error": "API error"}
        return response.json()
```

## 2. get\_sources

To retrieve sources from the API endpoint /sources



```
def get_sources(option, value):
    """
    getting the sources from endpoint /sources with the use of
    params to pass the API key and filtering criteria
    """
    params = {"apiKey": API_KEY, "pageSize": 15}
    if option == "category":
        params["category"] = value
    elif option == "country":
        params["country"] = value
    elif option == "language":
        params["language"] = value

    try:
        response = requests.get(f"{BASE_URL}/sources", params=params)
        if response.status_code != 200:
            print(f"Error with API: {response.status_code}, {response.text}")
            return {"error": "API error"}
        return response.json()

    except Exception:
        return {"API_error": "API error, check the connection"}
```

### 3. create\_file

This function will save the API responses into a file named with "<client\_name>

<group\_ID>.json"

```
def create_file(client_name, response, list, option):
    """
    ensure there is no space or special character in client name and file name
    and save the full API response into a json file
    """
    safe_client_name = re.sub(r"[^\w]", "_", client_name)
    file_name = (
        f"{safe_client_name}_{list.replace(' ', '_')}"
        f"-{option.replace(' ', '_')}_B4.json"
    )
    with open(file_name, "w", encoding="utf-8") as file:
        json.dump(response, file, ensure_ascii=False, indent=4)
```

### 4. prepare\_list

It will create a list containing only the information that the client will display to the user from the API response.

```
def prepare_list(response, list):  
    """  
    Taking the list of articles/sources and  
    prepare a list with only brief data for  
    only 15 elements  
    """  
    response = response[:15] # only 15 results  
    prepared_list = []  
  
    for x in response:  
        if list == "headlines":  
            prepared_list.append(  
                {  
                    "name": x["source"].get("name", "unknown"),  
                    "author": x.get("author", "unknown"),  
                    "title": x.get("title", "unkown"),  
                }  
            )  
        else:  
            prepared_list.append({"name": x.get("name", "unknown")})  
  
    return prepared_list
```

5. handle request After receiving a request, this fuction will handle it and return the full response list with source/headlines list

```
def handle_request(list, option, client_name, value=""):
    """
    Checking whether the client is looking for headlines or
    sources list and communicate with the appropriate function
    """
    if list == "headlines":
        if option == "all":
            print(f"client {client_name} requested to search for all {list}")
            full_response = get_all_headlines()
        else:
            print(
                f"client {client_name} requested to search for {list} by {option} ({value})"
            )
            full_response = get_headlines(option, value)
            response = full_response.get("articles", [])

    elif list == "sources":
        if option == "all":
            print(f"client {client_name} requested to search for all {list}")
        else:
            print(
                f"client {client_name} requested to search for {list} by {option} ({value})"
            )
            full_response = get_sources(option, value)
            response = full_response.get("sources", [])

    return response, full_response
```

6. search This function is responsible for receiving clients' requests and responding to them.

### Additional functions that used in both client and server side

- get\_local\_ip():

A function for retrieving the local IP of the device.

The IP address retrieved by this function is assigned for internal use(not public). It works by contacting 8.8.8.8 which is Google's public DNS server with a UDP socket where there is no need for sending data, this step is just to determine which network interface is being used.

After creating the socket it connects it to the DNS server. We can get the IP address using s.getsockname() which returns a tuple of the IP address and port number. s.getsockname()[0] will be the local IP address we need.

Finally, the method returns the IP address.

## Additional Concepts

**TLS\SSL (security)** TLS/SSL is a transport security protocol that provides a secure way for communication by providing some services. One of those services is confidentiality which depends on the encryption. On the screenshots below you can notice the data before applying TLS/SSL and after.

- Before TLS/SSL:

(1.)

The image shows a Wireshark packet capture of traffic on interface \Device\NPF\_{...}. The packet list shows a TCP segment (No. 78) from 192.168.0.121 to 192.168.0.121, Seq=1, Win=1731, Len=1731. The packet details pane shows the TCP segment data (380 bytes) and the payload (380 bytes). The packet bytes pane shows the raw data, which is unencrypted. A red box highlights the raw data, and a red text overlay states: "Data before TLS/SSL is not encrypted".

3. Quit  
Select an option: 1

----- Headlines menu -----  
1 - keywords  
2 - category  
3 - country  
4 - all  
5 - main

Select your option: 4

Headlines:  
No. (1)  
Source name: Wired  
author: Makena Kelly  
title: News Influencers Have Eclipsed Traditional Media

No. (2)  
Source name: [Removed]  
author: None  
title: [Removed]

No. (3)  
Source name: [Removed]  
author: None  
title: [Removed]

No. (4)  
Source name: Gizmodo.com  
author: Matthew Gault  
title: Matt Gaetz Hates Big Tech (That's the Good News)

No. (5)  
Source name: Gizmodo.com  
author: Matt Novak  
title: Popular Tech News Site The Verge Goes Behind a Paywall

(2.)

The image shows a Wireshark packet capture of traffic on interface \Device\NPF\_{...}. The packet list shows a TCP segment (No. 18) from 192.168.0.144 to 192.168.0.144, Seq=18, Win=326912, Len=380. The packet details pane shows the TCP segment data (380 bytes) and the payload (380 bytes). The packet bytes pane shows the raw data, which is unencrypted. A red box highlights the raw data, and a red text overlay states: "Data before TLS/SSL is not encrypted".

----- Sources menu -----  
1 - category  
2 - country  
3 - language  
4 - all  
5 - main

Select your option: 4

Sources:  
No. (1)  
Source name: ABC News

No. (2)  
Source name: ABC News (AU)

No. (3)  
Source name: Aftenposten

No. (4)  
Source name: Al Jazeera English

No. (5)  
Source name: ANSA.it

No. (6)  
Source name: Angam

No. (7)  
Source name: Ars Technica

No. (8)  
Source name: Ary News

No. (9)

- After TLS/SSL:



(1.)

The image shows a Wireshark packet capture of traffic on interface \Device\NPF\_{...}\_loopback, id 0. The filter is 'tcp.port == 5353'. The packet list shows a series of TCP segments. Packet 10 is selected, showing details for Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The packet bytes pane shows the raw data, with a red box highlighting the encrypted data (application/javascript). The packet details pane shows the Hypertext Transfer Protocol section, indicating that the data is encrypted.

Wireshark - Packet 10 - Adapter for loopback traffic capture

Frame 10: 299 bytes on wire (2392 bits), 299 bytes captured (2392 bits) on interface \Device\NPF\_{...}\_loopback, id 0

Null/Loopback

Internet Protocol Version 4, Src: 192.168.0.121, Dst: 192.168.0.121

Transmission Control Protocol, Src Port: 5353, Dst Port: 47471, Seq: 1450, Ack: 598, Len: 255

Source Port: 5353

Destination Port: 47471

[Stream index: 0]

[Conversation completeness: Incomplete, DATA (15)]

... .. = RST: Absent

... .. = FIN: Absent

... .. = Data: Present

... .. = ACK: Present

0000 fa 94 0a 55 ff 1b 3e 52 14 31 03 c7 52 01 df

0001 83 45 f6 8b da e5 f5 aa e1 7e 93 7f 83 e3 19

0002 e2 d0 71 c3 74 17 7c a5 3c d0 b2 da cf df 45

0003 a6 10 a3 36 5f 81 dc 24 6e 2a f8 62 6d f8 bc

0004 e8 0c 4b a8 ba b6 66 f4 9a 3f 60 ad 5b 12

0005 a5 19 0b 06 85 d0 64 c0 f9 dc 74 06 b0 f1 32

0006 e6 06 70 22 54 84 0c bc 42 3e a2 e8 ed 79 19

0007 f7 75 57 10 33 20 cf 8e 36 a2 f0 22 f8 44 35

0008 10 09 30 07 a4 20 c7 53 92 92 3c 2a 70 b1 92

0009 13 ab b7 be 06 79 b8 f2 ec 65 ad 5a 71 31 11

000a 23 3c be 4e 46 df d1 c4 13 f2 ea b3 5c 6e 06

000b 03 43 7d 9c a2 e2 c1 e6 57 a8 47 58 ad e1 a7

000c 5b 4d 35 88 33 b1 29 1a 49 ef 23 7f 55 2c eb

000d 64 59 00 ef 03 13 1d 92 2d 13 c9 42 60 c5 50

000e 1d bd d5 bb 4e bf e3 8c 4d 34 b2 97 eb 0b 86

No: 10 - Time: 0.014403 - Source: 192.168.0.121 - Destination: 192.168.0.121 - Protocol: TCP - Seq: 1450 - Ack: 598 - Len: 255 [TCP segment of a reassembled PDU]

Show packet bytes

Profile: Default

(2.)

The image shows a Wireshark packet capture of traffic on interface \Device\NPF\_{...}\_loopback, id 0. The filter is 'tcp.port == 5353'. The packet list shows a series of TCP segments. Packet 18 is selected, showing details for Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The packet bytes pane shows the raw data, with a red box highlighting the encrypted data (application/javascript). The packet details pane shows the Hypertext Transfer Protocol section, indicating that the data is encrypted.

Wireshark - Packet 18 - Adapter for loopback traffic capture

Frame 18: 446 bytes on wire (3568 bits), 446 bytes captured (3568 bits) on interface \Device\NPF\_{...}\_loopback, id 0

Null/Loopback

Internet Protocol Version 4, Src: 192.168.0.144, Dst: 192.168.0.144

Transmission Control Protocol, Src Port: 5353, Dst Port: 7044, Seq: 1544, Ack: 659, Len: 402

Source Port: 5353

Destination Port: 7044

[Stream index: 0]

[Conversation completeness: Incomplete, DATA (15)]

... .. = RST: Absent

... .. = FIN: Absent

... .. = Data: Present

... .. = ACK: Present

0000 02 00 00 00 45 00 01 5a de 25 40 00 00 06 00 00

0001 c0 a8 00 90 c0 a8 00 90 14 e9 1b 84 dc 6c fa e4

0002 7f f9 a2 68 50 18 20 f8 9b 2c 00 00 17 03 03 05

0003 8d c0 1e 75 24 a1 02 2e 7f 3f 01 50 fa 00 2d a2

0004 6a c0 34 b3 1b 5d c6 94 c6 bd 59 af 80 1e f4 d7

0005 4d 34 fc 90 c8 2f bc d7 06 ae 5c 27 78 82 bd 95

0006 34 38 5d c0 04 b1 a1 ea 1c ae c5 e9 99 f2 3a 40

0007 d6 6b b0 67 9d 47 fb c8 fa c4 f1 a9 57 ed 7a e3

0008 8b 1b 59 07 5a cb 3b 35 ee c2 cc ad a1 f2 70 c4

0009 48 0b 74 85 57 1a 93 a5 f0 6f 5b 46 82 00 3c ed

000a a6 88 f9 38 ec e8 2e 59 c0 40 4b 53 8d c8 12 3b

000b 8c ca 46 70 e9 5b 68 c9 3c ea 59 cf a4 c2 9e 83

000c 95 d6 68 48 c3 57 fc 1a 3f 2b 62 8a 1c ea b8 d7

000d bb 92 23 a0 82 5d 13 85 64 87 42 4c 56 19 91

000e c2 00 88 6b b4 42 8d 80 6e b0 cd 00 10 98 b4 95

No: 18 - Time: 11.544191 - Source: 192.168.0.144 - Destination: 192.168.0.144 - Protocol: TCP - Seq: 1544 - Ack: 659 - Len: 402 [TCP segment of a reassembled PDU]

Show packet bytes

Profile: Default

Acknowledgments

We would like to thank our instructor for providing this project so we can learn how to implement a python network system. Moreover, a big thanks to NewsAPI for providing the news.

## Conclusion

This project demonstrates how secure connections, API integration, Sockets, and client-server communication can be practically implemented using a Python network application.

The development of the Multithreaded News Client/Server Information System helped us learn concepts of:

- Network programming (Python)
- API integration
- Multi-threaded
- client\server framework

## Resources

- API: <https://newsapi.org/>
- TLS/SSL: <https://docs.python.org/3/library/ssl.html>