Lab 2

Distributed Software Systems

UNIBO - 2024/25

Anouk Wuyts - 1900124167

Davide De Rosa - 1186536

Simple Remote File Reader

Goal: allow a client to read files from a remote server as if they were local files, without the client needing to know the file is on a remote machine.

Challenge: achieve *network transparency,* where users shouldn't know where the file they want to read is located.

Network Transparency

Definition: *network transparency* refers to the ability of the system to hide the complexities and details of the network from the users.

This means that users interact with the system as though it were a *single cohesive entity*, rather than a collection of separate, networked machines.

The underlying network communications, data transfers, and resource access across different nodes (machines) are invisible to the user, providing a seamless experience.

Network transparency aims to make distributed resources and services appear as local resources, minimizing the user's awareness of the distributed nature of the system.

Goals of Network Transparency

Location Transparency: ensures that users or applications do not need to know the physical location of resources or services. They interact with resources as if they are all local, even if they are distributed across different machines or geographic locations.

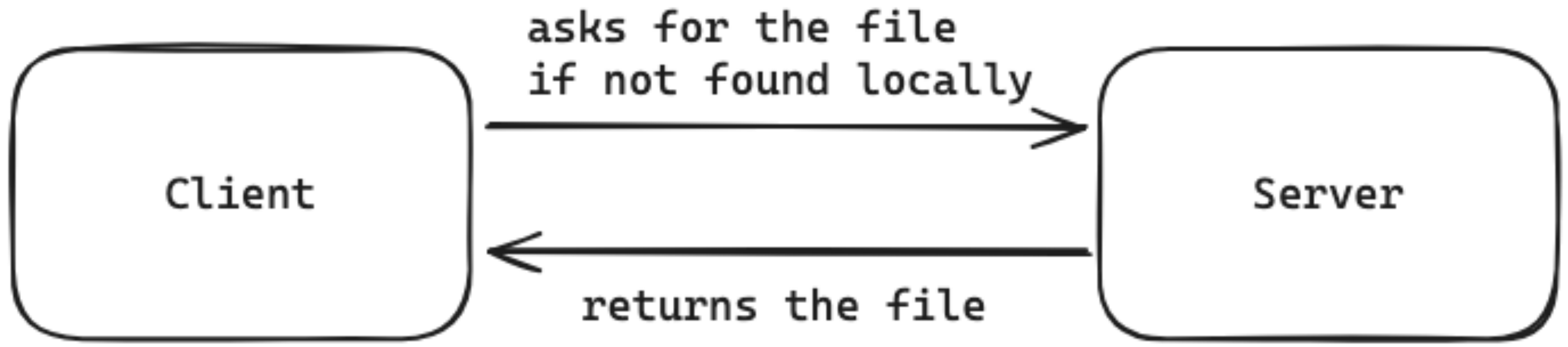
Access Transparency: ensures that the method of accessing resources is consistent, regardless of where they are located or how they are accessed (locally or remotely). Users should not need to know if the resource is on the local machine or elsewhere, nor worry about differences in access methods.

Failure Transparency: the system should mask network or node failures, allowing users to continue interacting with the system without noticing that a failure occurred. The system should recover or retry operations automatically, providing a robust and resilient experience.

Implementation

We choose to implement the solution by using **Flask** for the **server**, exposing an *API* that asks for a *filename* as a query parameter and returns a file if the filename is found inside the server files directory.

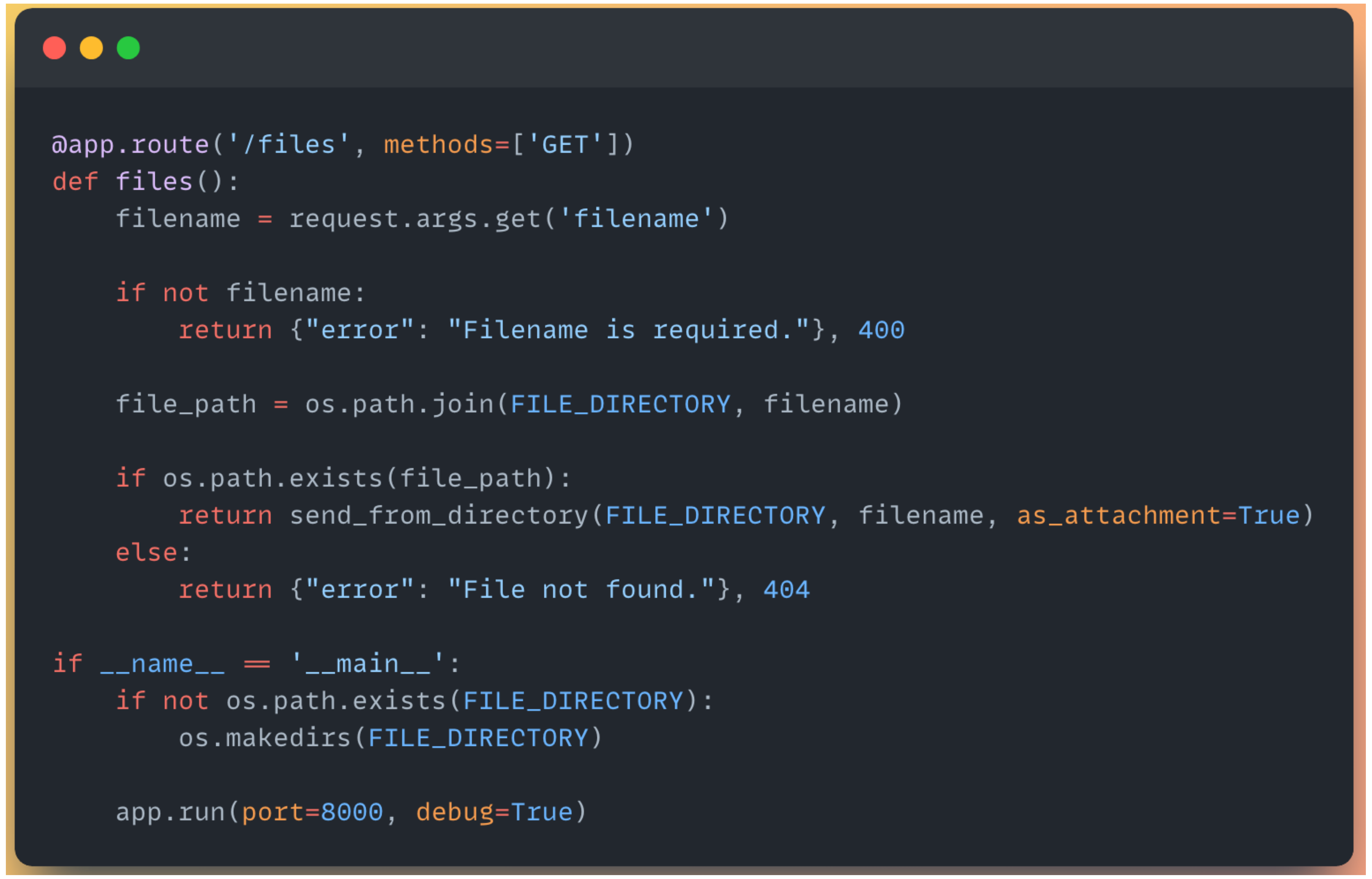
As for the **client**, after asking for the filename, we check if the file is already located on the local machine. Otherwise, an API request to the server is going to get the file and save it locally. The file will be printed on the console, showing the text to the user.



Server

Using **Flask**, we expose a *GET HTTP method*, which asks for a filename in the query and gives back a file.

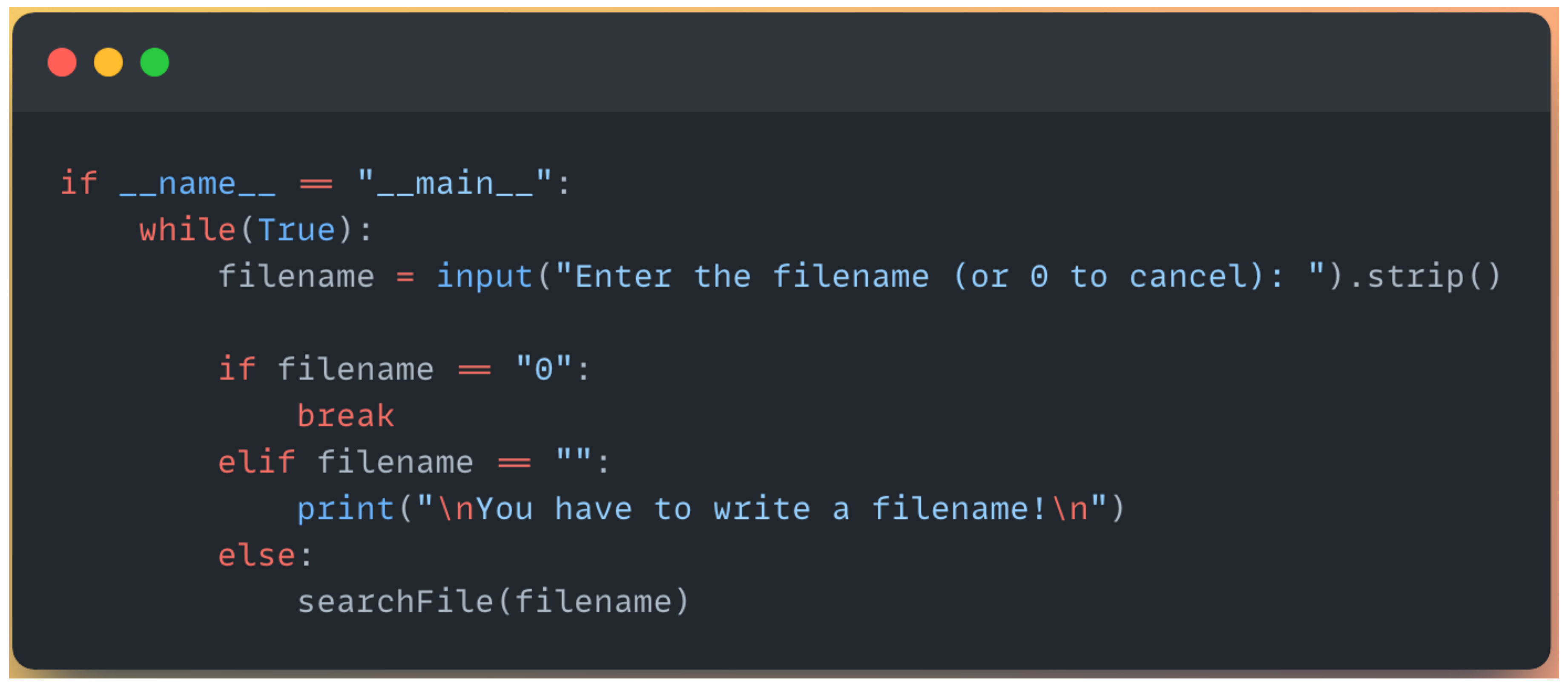
If the filename is null or if the file is not found on the server directory, it returns an error message.



```python
@app.route('/files', methods=['GET'])
def files():
    filename = request.args.get('filename')

    if not filename:
        return {"error": "Filename is required."}, 400

    file_path = os.path.join(FILE_DIRECTORY, filename)

    if os.path.exists(file_path):
        return send_from_directory(FILE_DIRECTORY, filename, as_attachment=True)
    else:
        return {"error": "File not found."}, 404

if __name__ == '__main__':
    if not os.path.exists(FILE_DIRECTORY):
        os.makedirs(FILE_DIRECTORY)

    app.run(port=8000, debug=True)
```

Client

The *client* continuously asks the user what he wants to do. When he writes a filename, the *searchFile()* function is called.

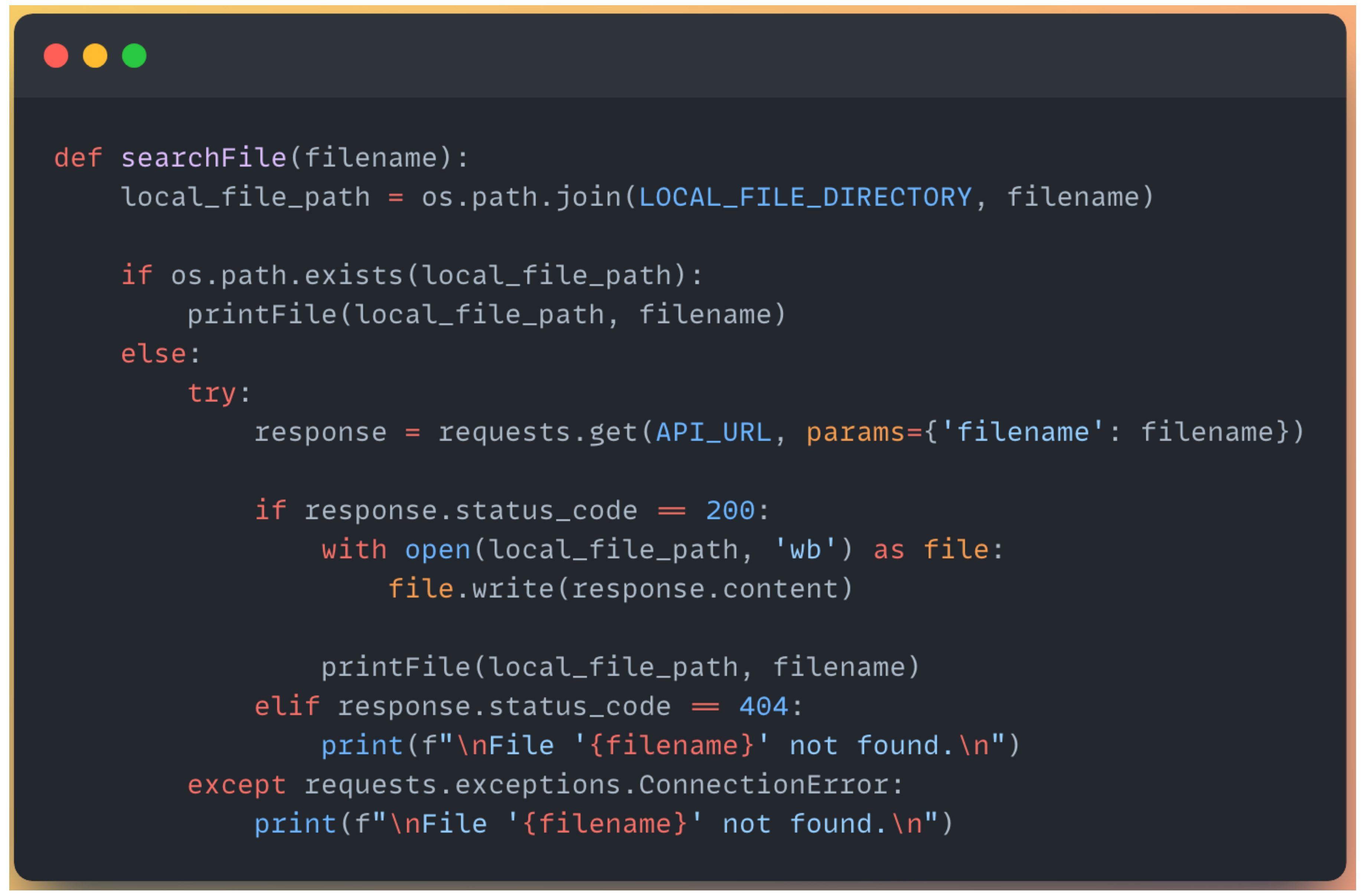


```python
if __name__ == "__main__":
    while(True):
        filename = input("Enter the filename (or 0 to cancel): ").strip()

        if filename == "0":
            break
        elif filename == "":
            print("\nYou have to write a filename!\n")
        else:
            searchFile(filename)
```

searchFile() looks for the given filename locally. If the files is found, the *printFile()* function is used.

Otherwise, the client makes the API call to the server to get the file. If the file is found on the server, it gets returned, saved locally, and printed on the console via the same function.

In the case that the server is not running, the client will handle that exception by printing that the file has not been found.

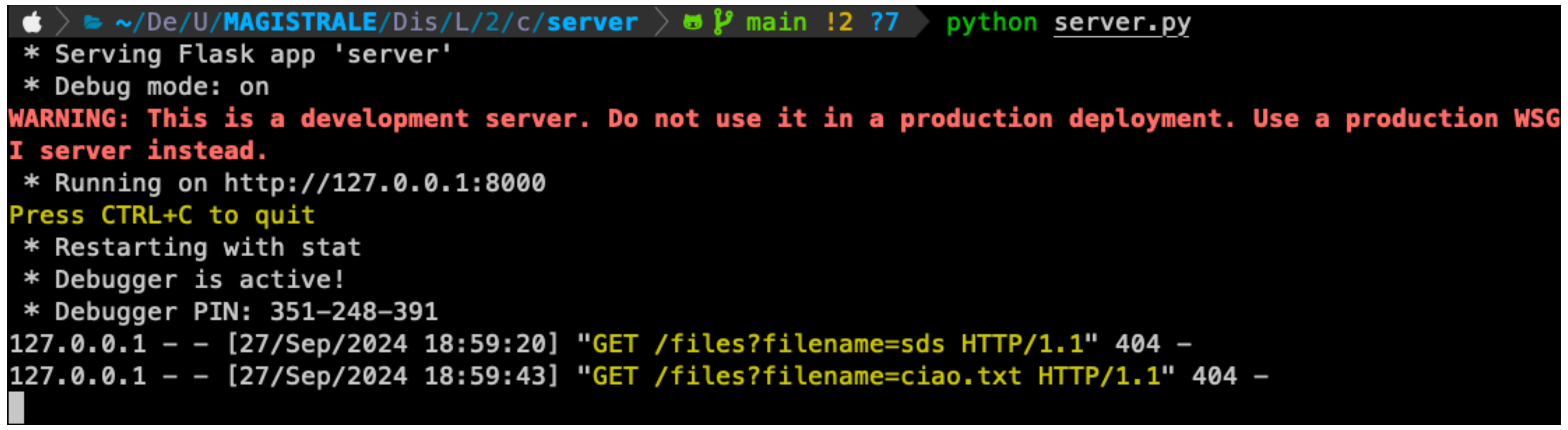


```python
def searchFile(filename):
    local_file_path = os.path.join(LOCAL_FILE_DIRECTORY, filename)

    if os.path.exists(local_file_path):
        printFile(local_file_path, filename)
    else:
        try:
            response = requests.get(API_URL, params={'filename': filename})

            if response.status_code == 200:
                with open(local_file_path, 'wb') as file:
                    file.write(response.content)

                printFile(local_file_path, filename)
            elif response.status_code == 404:
                print(f"\nFile '{filename}' not found.\n")
        except requests.exceptions.ConnectionError:
            print(f"\nFile '{filename}' not found.\n")
```

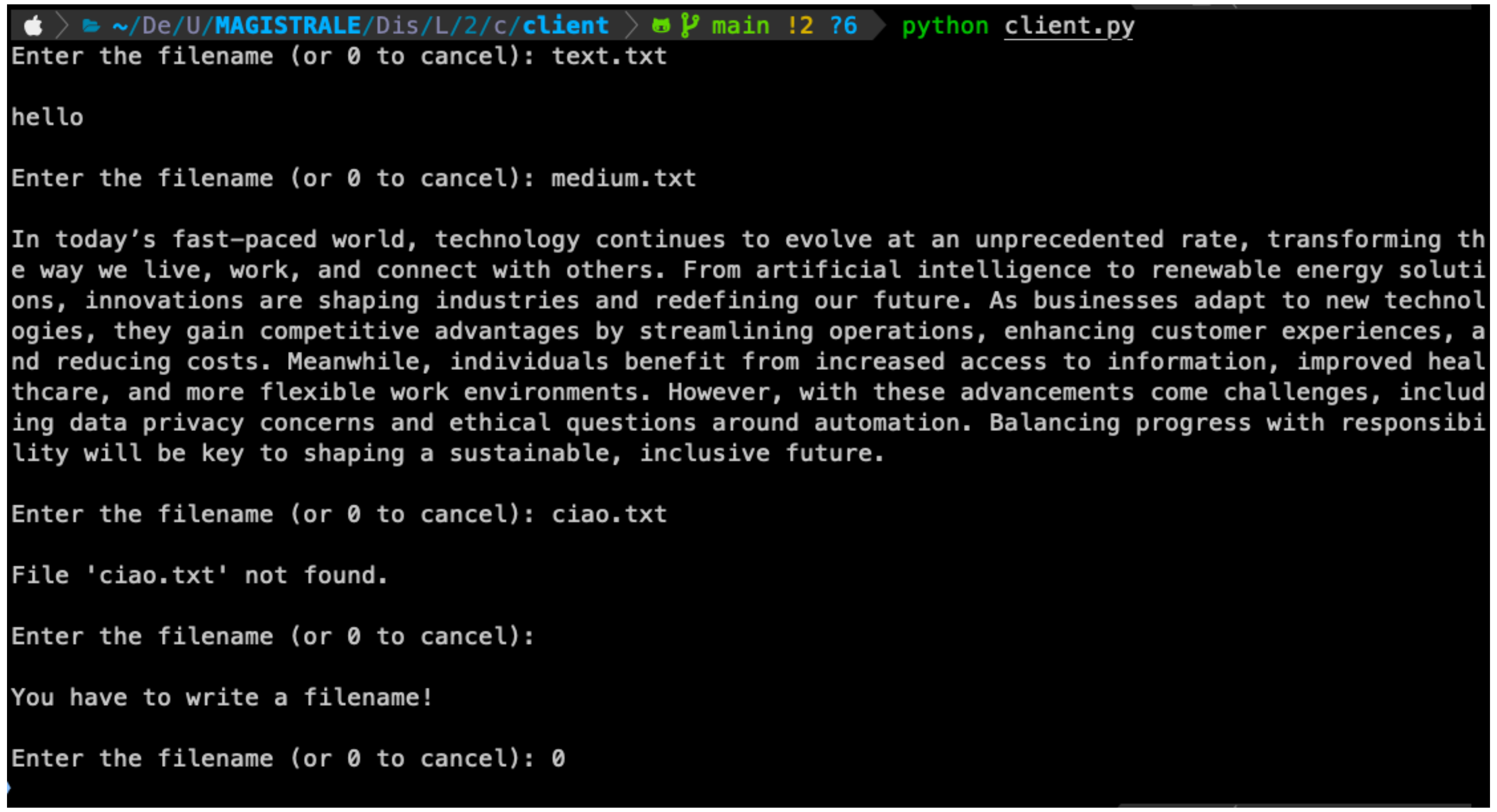
Usage

After executing the server code, you can run the *client.py* file, which asks you for the filename. We choose to leave the extension open for the user to write, leaving the possibility to them to open different types of text files.

The user can't notice the difference between a locally or remotely saved file, ensuring *network transparency*. When the file is not found locally, the server returns the file which is saved locally and printed on the console.



```
 ~/De/U/MAGISTRALE/Dis/L/2/c/server  main !2 ?7  python server.py
 * Serving Flask app 'server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSG
I server instead.
 * Running on http://127.0.0.1:8000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 351-248-391
127.0.0.1 - - [27/Sep/2024 18:59:20] "GET /files?filename=sds HTTP/1.1" 404 -
127.0.0.1 - - [27/Sep/2024 18:59:43] "GET /files?filename=ciao.txt HTTP/1.1" 404 -
```

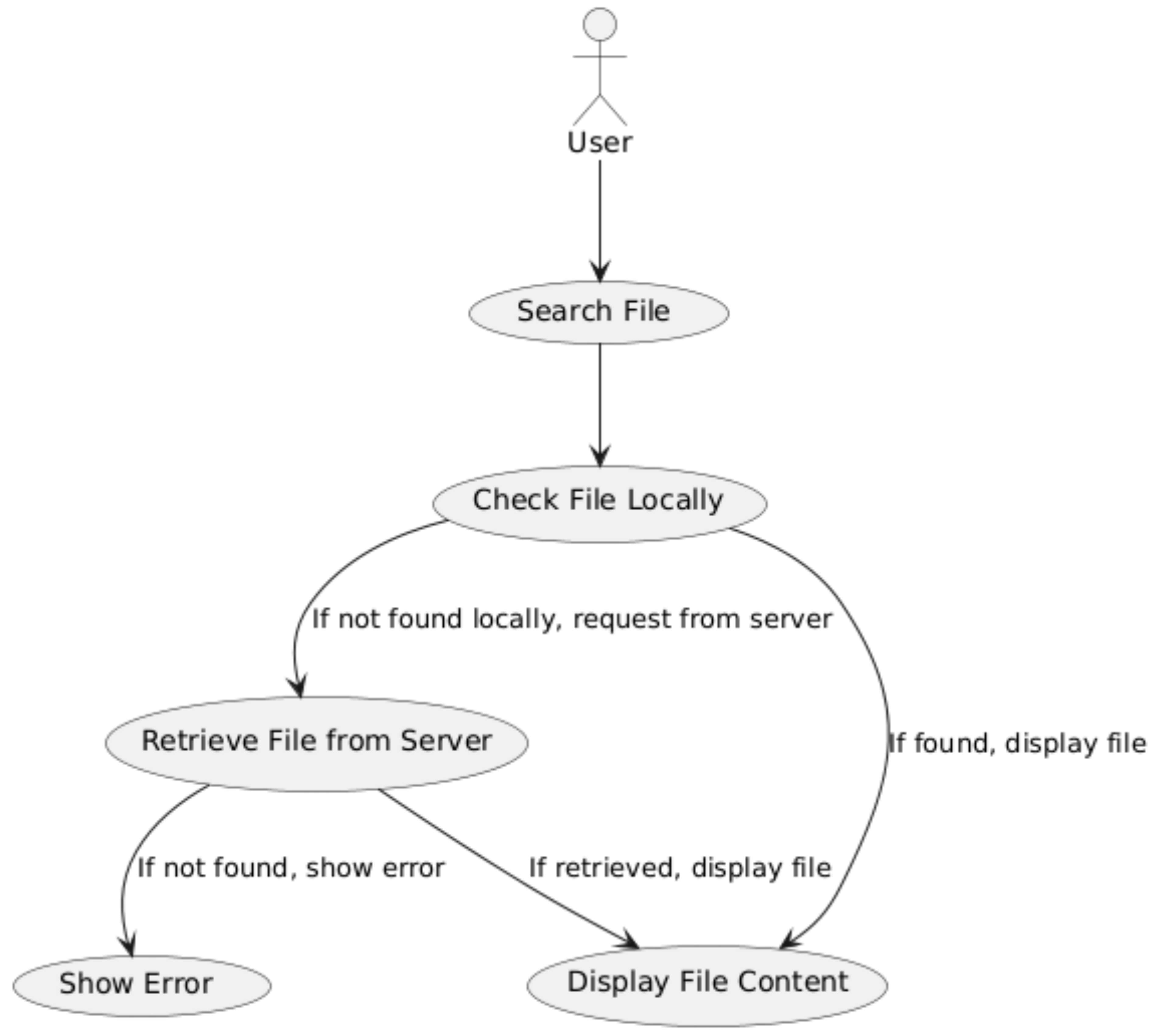


```
 ~/De/U/MAGISTRALE/Dis/L/2/c/client  main !2 ?6  python client.py
Enter the filename (or 0 to cancel): text.txt

hello

Enter the filename (or 0 to cancel): medium.txt

In today's fast-paced world, technology continues to evolve at an unprecedented rate, transforming th
e way we live, work, and connect with others. From artificial intelligence to renewable energy soluti
ons, innovations are shaping industries and redefining our future. As businesses adapt to new technol
ogies, they gain competitive advantages by streamlining operations, enhancing customer experiences, a
nd reducing costs. Meanwhile, individuals benefit from increased access to information, improved heal
thcare, and more flexible work environments. However, with these advancements come challenges, includ
ing data privacy concerns and ethical questions around automation. Balancing progress with responsibi
lity will be key to shaping a sustainable, inclusive future.

Enter the filename (or 0 to cancel): ciao.txt

File 'ciao.txt' not found.

Enter the filename (or 0 to cancel):

You have to write a filename!

Enter the filename (or 0 to cancel): 0
```

Usage - UML

We also used an *UML graph* to look at the client algorithm in a different way.

With this graph, its clearly visible how the client works and how he handles every step of the algorithm.



Thank you!