Network Communication Technologies

Multi-User Real-Time Text Editor

Sumeyra Karsavran

June 2025

1. INTRODUCTION

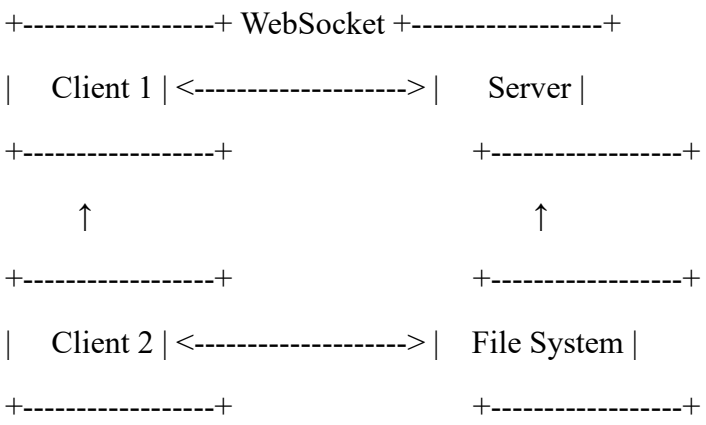
Objective:
In this project, a system was developed that allows multiple users to work on text files simultaneously. The system consists of a WebSocket server and tkinter GUI-based clients.

Technologies:

- Python
- websockets (WebSocket protocol)
- tkinter (GUI)
- Asyncio (Asynchronous Programming)
- JSON (data exchange)

2. SYSTEM ARCHITECTURE

2.1 General Structure:



2.2 Components:

Component	Explanation
Server	It manages all files. Saves and publishes changes.
Client	It offers an interface to the user. It provides operations such as file creation and editing.
File System	Files are stored in the server_files/ directory. The server writes the file to disk with each update.

3. SERVER APPLICATION

Key Duties:

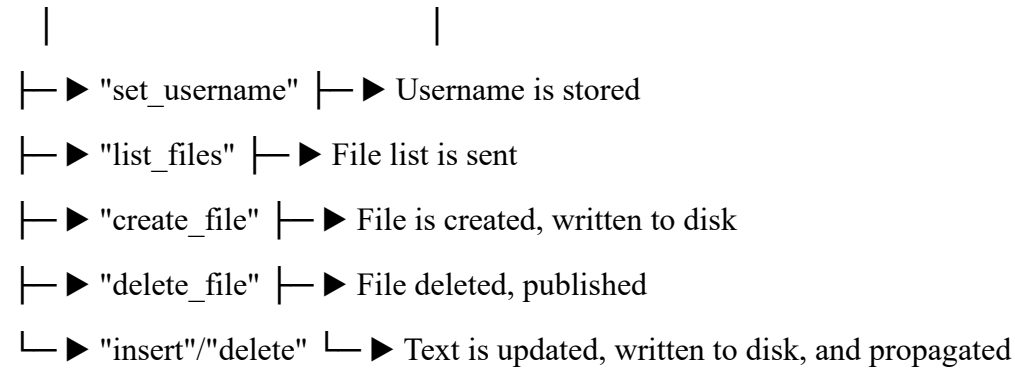
- Listens for WebSocket connections
- Creates, deletes and lists files
- Publishes changes made to a file to all users
- Save/upload files to the server directory

Data Structures:

Variable	Explanation
clients	Maintains WebSocket connections and each client's open file/username
files	File names and contents are stored as a dictionary

Operation Diagram (Server):

Client sends message ──▶ JSON is resolved ──▶



4. CLIENT APP

Key Duties:

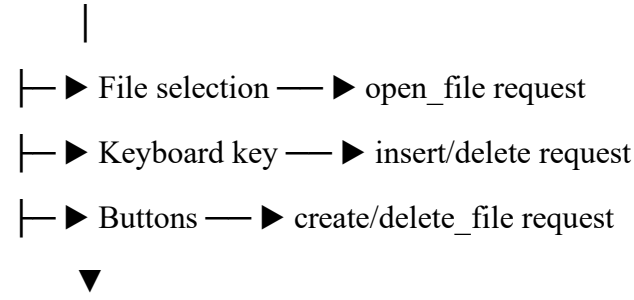
- Retrieves the username from the user
- Shows the list of files
- Shows the contents of the selected file
- Notifies the server of keyboard changes made
- Takes the changes made by other users and reflects them in the application

Interface Features:

Component	Function
Listbox	Shows the list of files on the server
Text	Shows the contents of the active file and allows editing
Button	Used for file creation and deletion

Functioning Diagram (Client):

User interaction ──▶ Event Handler (GUI)



Incoming messages are asynced and transferred to the GUI thread

5. MESSAGE STRUCTURES (JSON)

Action	Sender	Area	Explanation
set_username	Client	Server	Reports username
list_files	Client	Server	Requests the file list
files_list	Server	Client	Sends existing files
create_file	Client	Server	Creates a new file
delete_file	Client	Server	File deletion request
open_file	Client	Server	Requests file contents
file_content	Server	Client	Sends file contents
insert	Client	Server & Other Clients	Instructs the addition of characters to the content
Delete	Client	Server & Other Clients	Deletes specific characters

6. OPERATING PRINCIPLES

6.1 States

State	Explanation
Disconnected	WebSocket connection not yet established
Connected	WebSocket connection established, username not sent
LoggedIn	Username submitted, server file list may return
FileSelected	A file is selected, its contents are retrieved
Editing	User editing text, submitting changes

6.2 Transitions

Event	Source Status	Target Situation	Explanation
connect()	Disconnected	Connected	WebSocket connection established
send_username()	Connected	LoggedIn	Username submitted
list_files()	LoggedIn	LoggedIn	File list retrieved from server
select_file()	LoggedIn	FileSelected	File selected, content retrieved
edit_text()	FileSelected	Editing	User added/deleted text
receive_edit()	FileSelected/Editing	Editing	Changes made by other users were taken
disconnect()	*Any State*	Disconnected	Connection broken/closed

6.3 FSM Diagram

6.4 FSM Mermaid Notation

```
stateDiagram-v2
[*] --> Disconnected
Disconnected --> Connected: connect()
Connected --> LoggedIn: send_username()
LoggedIn --> LoggedIn: list_files()
LoggedIn --> FileSelected: select_file()
FileSelected --> Editing: edit_text()
Editing --> Editing: receive_edit()
FileSelected --> Editing: receive_edit()
Disconnected <--> [*]: disconnect()
```

6.5 Protocol Message Types and Transaction Table

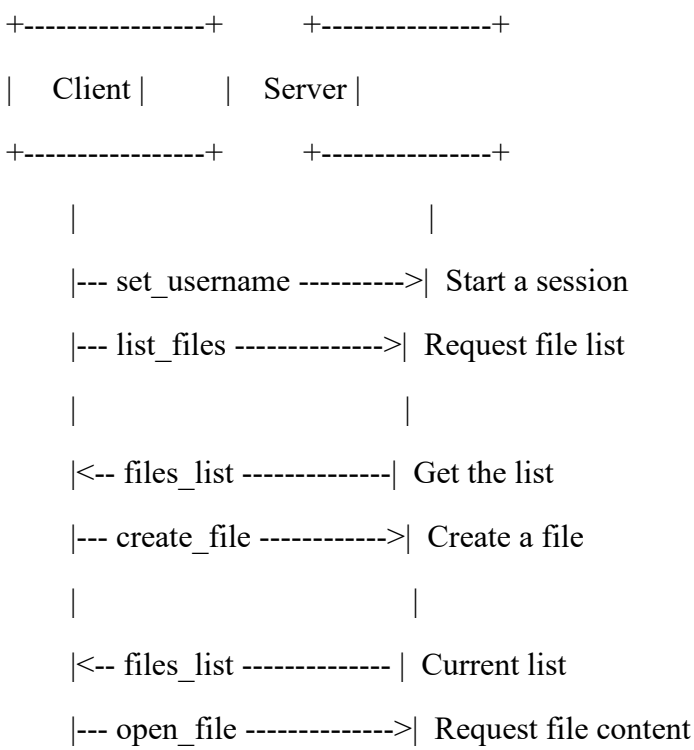
Message Type (action)	**Sender**	**Buyer**	**Message Content**	**Buyer's Action**
set_username	Client	Server	{ "action": "set_username", "username": "John" }	Defines the user to the clients dictionary.
list_files	Client	Server	{ "action": "list_files" }	files_list returns the list of files with the message.
files_list	Server	Client	{ "action": "files_list", "files": ["a.txt", "b.txt"] }	Updates the list box on the interface.
create_file	Client	Server	{ "action": "create_file", "filename": "yeni.txt" }	files list, writes to disk, files_list paths to all clients.

Message Type (action)	**Sender**	**Buyer**	**Message Content**	**Buyer's Action**
delete_file	Client	Server	{ "action": "delete_file", "filename": "sil.txt" }	files list, removes the file on the disk, publishes files_list message.
open_file	Client	Server	{ "action": "open_file", "filename": "a.txt" }	It finds the file contents, sends file_content message to the client.
file_content	Server	Client	{ "action": "file_content", "filename": "a.txt", "content": "..." }	Populates the Text field with incoming content.
insert	Client	Server	{ "action": "insert", "filename": "a.txt", "index": "2.5", "content": "a" }	Adds content to the specified location, saves the file, sends the message to other clients.
Delete	Client	Server	{ "action": "delete", "filename": "a.txt", "index": "2.5", "length": 1 }	Deletes the character(s) in the specified location, saves them, sends them to other clients.
insert	Server	Client	{ "action": "insert", "filename": "a.txt", "index": "2.5", "content": "a" }	If the open file matches, it adds to the client's interface.
Delete	Server	Client	{ "action": "delete", "filename": "a.txt", "index": "2.5", "length": 1 }	If the open file matches, the client deletes the corresponding character(s).
ConnectionClosed	-	-	-	Server: removes from the list of clients. Client: The GUI continues to work, but the connection is lost.

6.6 Basic Principle of Compatible Communication of Two End Systems Without Knowing Coding

1. The parties do not know how each other works.
    - For example, the client doesn't know how the server stores files.
    - The server also does not know how the client's GUI is drawn.
2. They both only implement the protocol.
    - The client sends a "create_file" message.
    - The server only checks that the message is "action": "create_file" and takes the necessary action.
3. Mutual understanding guarantee:
    - Thanks to the use of JSON, every domain ("filename", "content") is open and fixed.
    - Both parties act according to these areas and do not comment on what the data in the content is.

6.7 Protocol Flow: Diagram Symbol Representation



|<-- file_content ------------| Get content

| |

|--- insert / delete -------->| Make arrangements

|<-- insert / delete ---------| Spread to other clients

| |

|--- delete_file ------------>| Delete file

|<-- files_list --------------| Current list

7. SECURITY AND RESTRICTIONS

- Users can't see each other's IDs (only content syncs)
- There may be some race cases in simultaneous writing (line-based analysis can be applied)

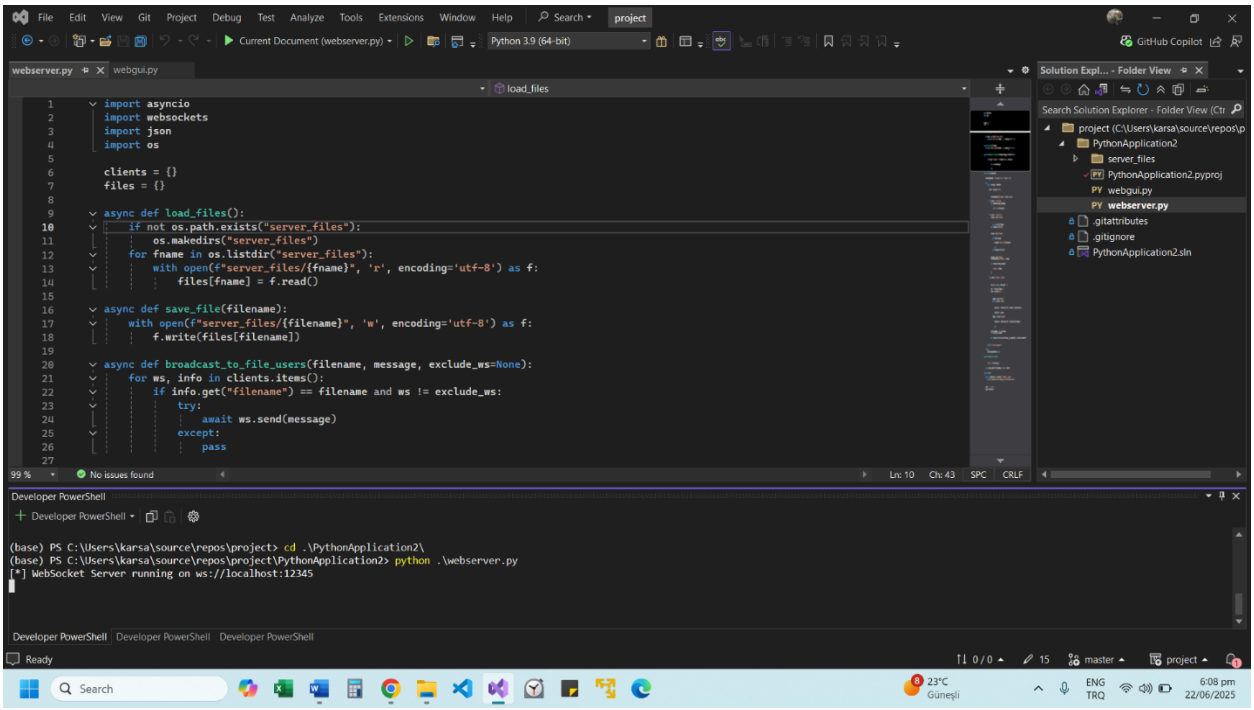
8. DEVELOPMENT SUGGESTIONS

- **Username Representation:** Username can be displayed in every edit.
- **Versioning:** File versions can be stored.
- **Access Authorizations:** Special file permissions can be added to each user.

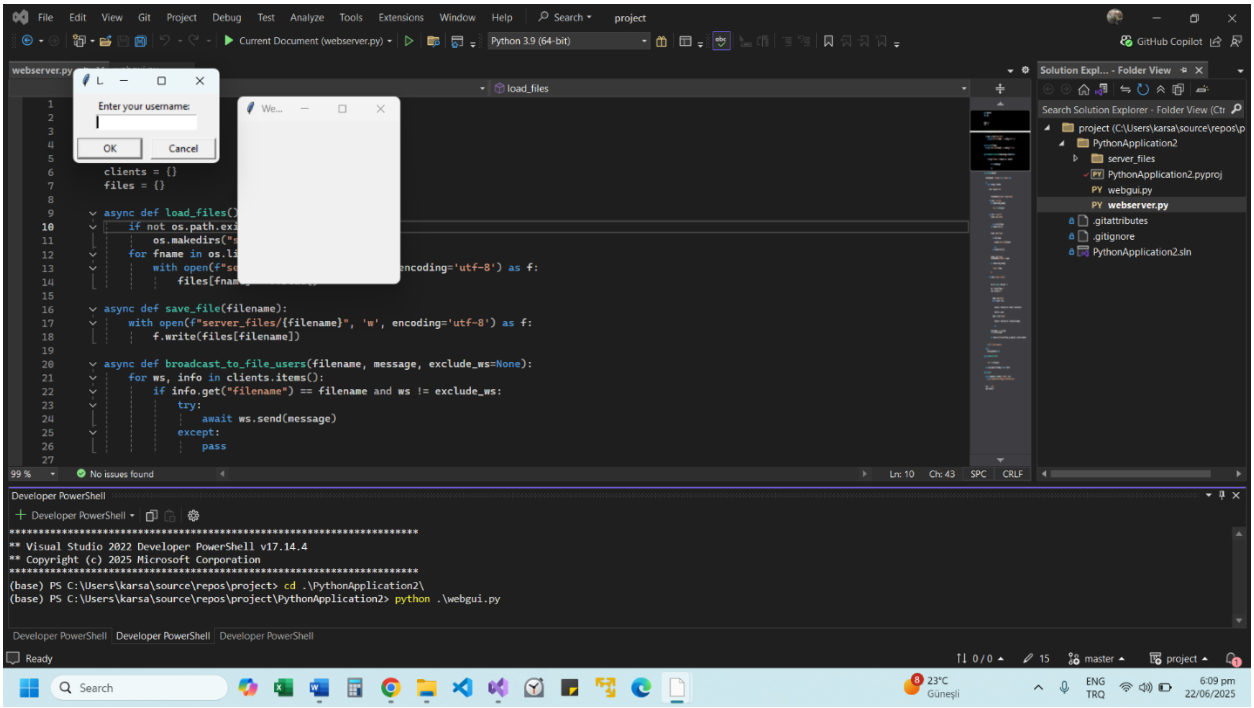
9. CONCLUSION

This project is very fruitful in terms of experiencing the process of developing a real-time collaboration application with WebSocket and demonstrating how technologies such as asyncio and tkinter can work together. It is a structure that can form the basis for scenarios such as different users working on a document simultaneously.

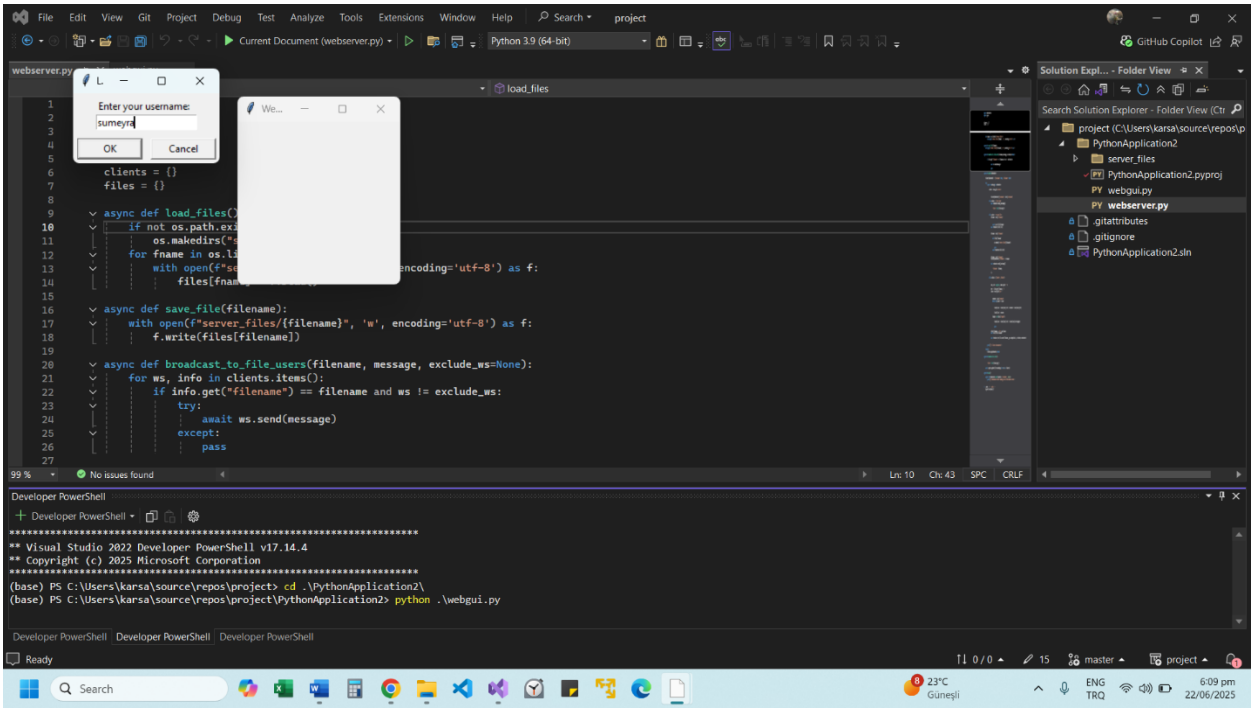
10. SCREENSHOTS & PRINTOUTS



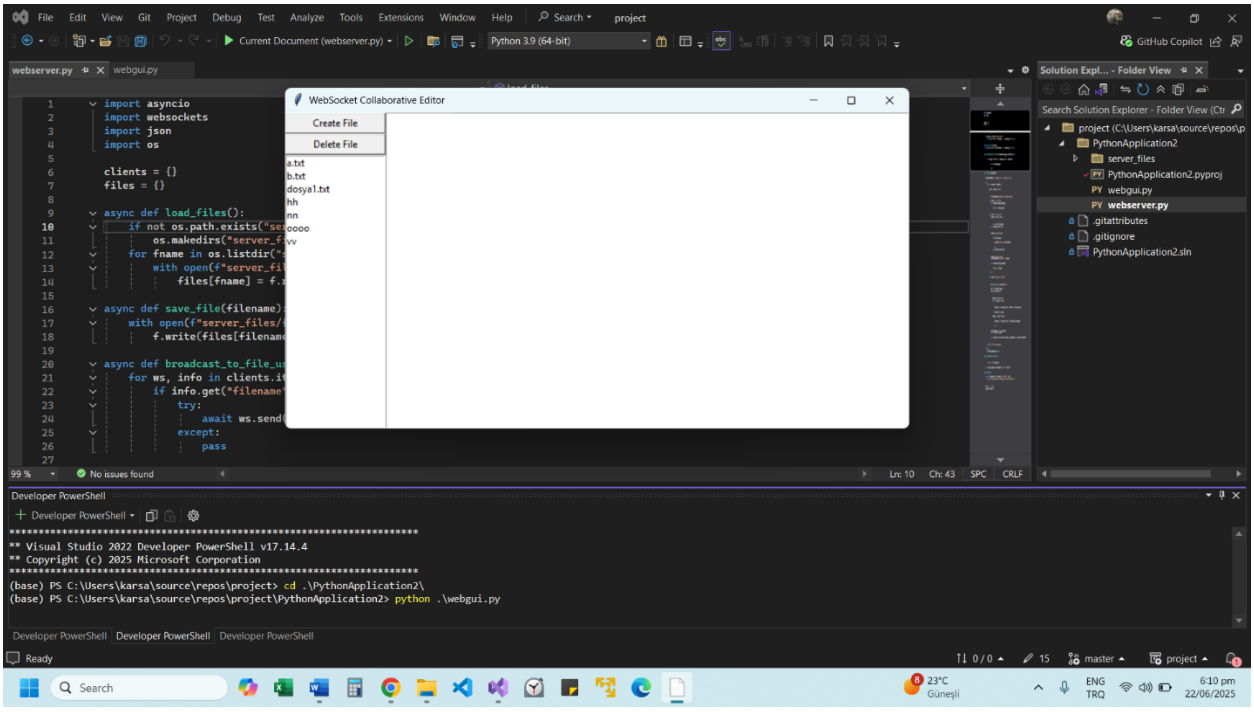
Connecting to the server



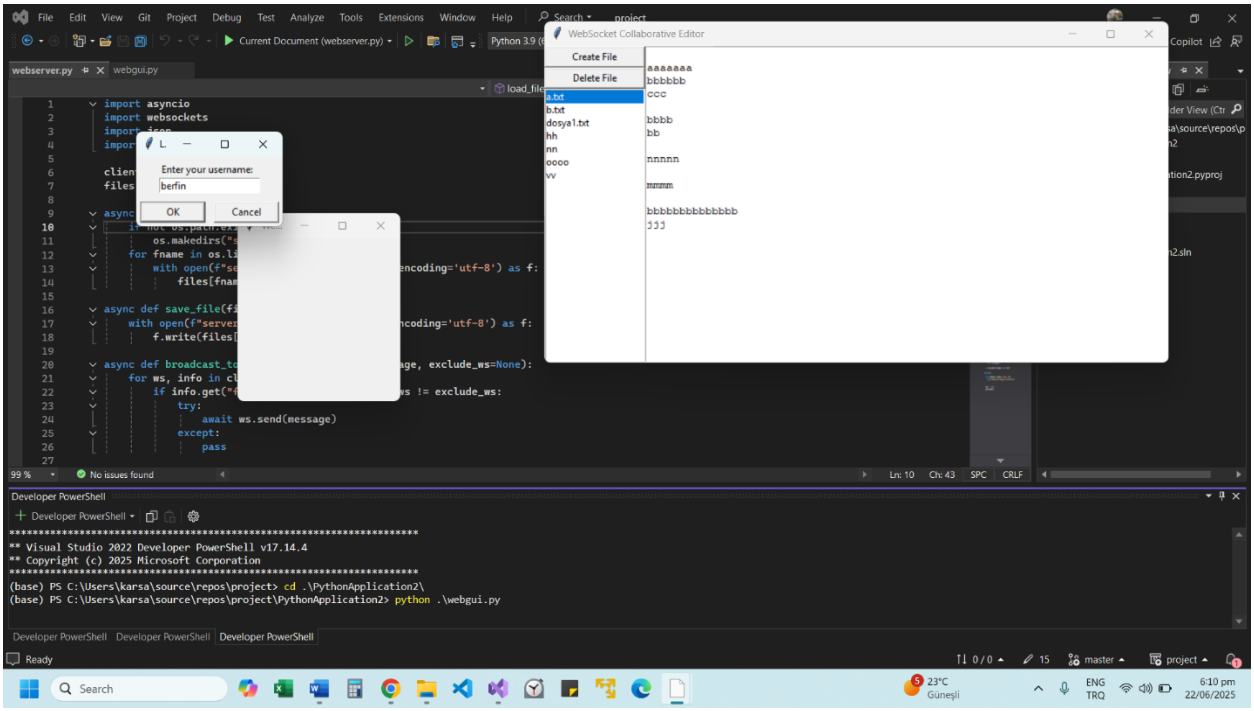
The screens that appear after executing the client code are shown. When you type a username, the text editor screen is active.



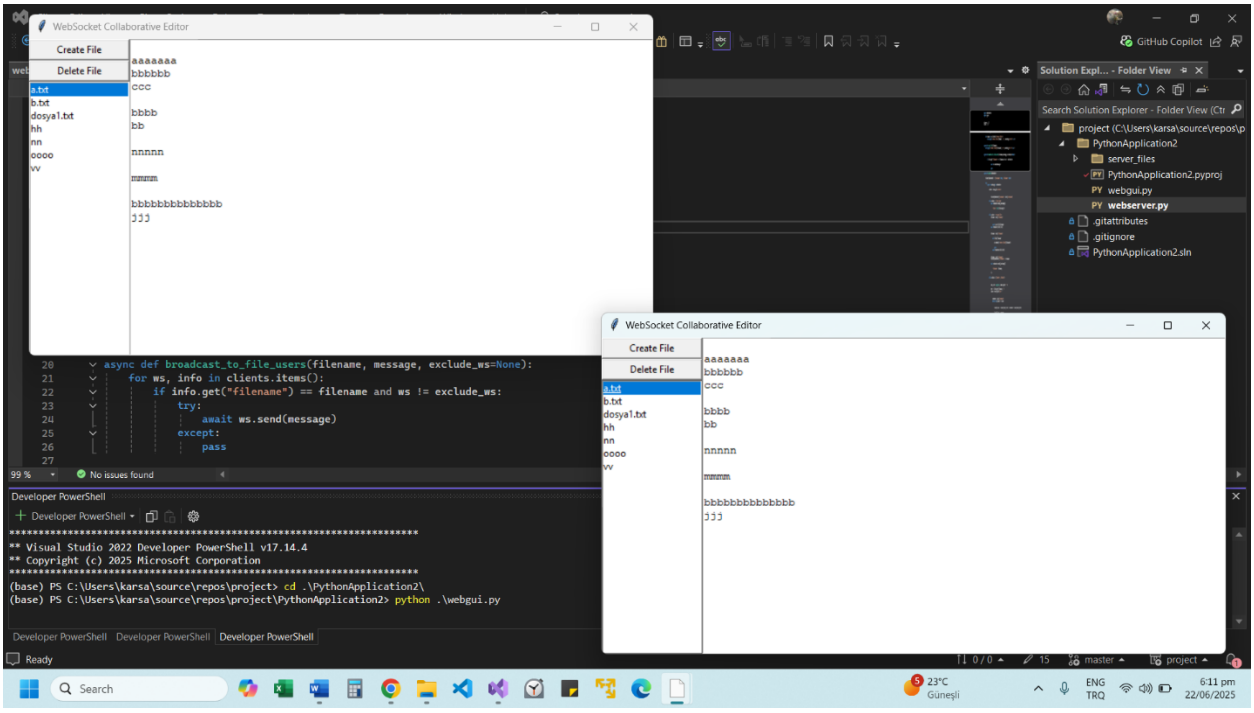
Enter the username and click on the arrow.



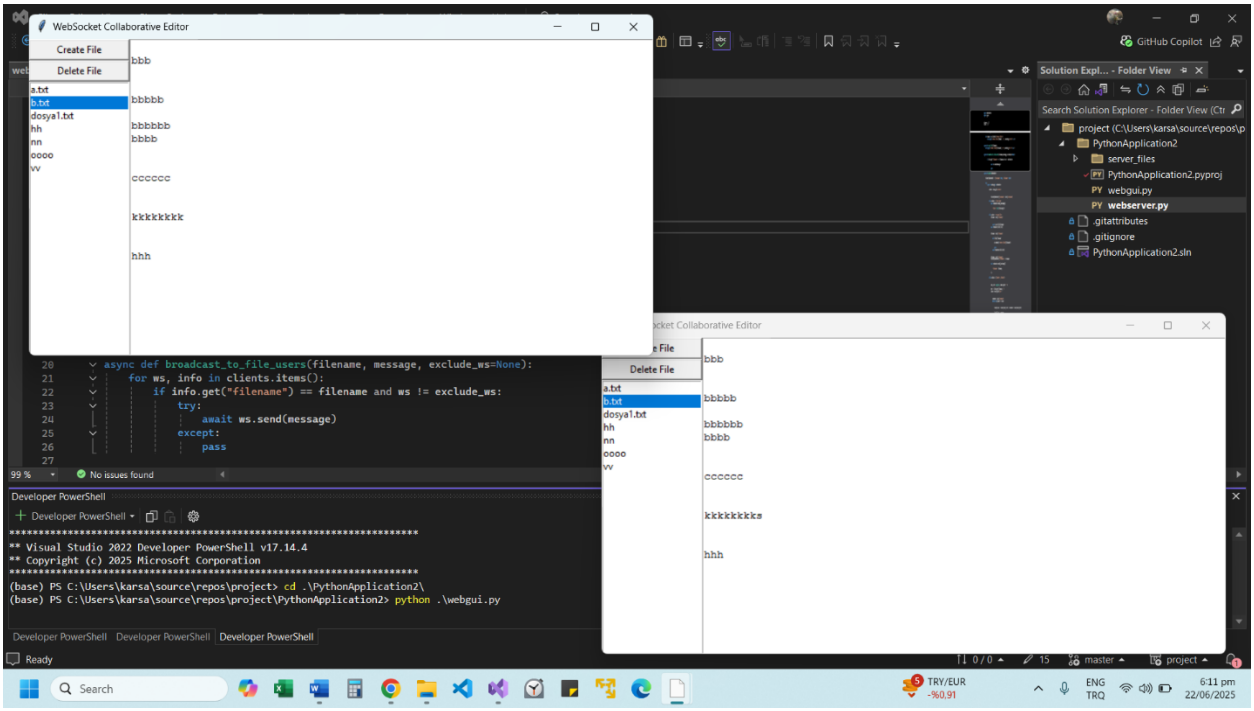
The text editor has been activated.



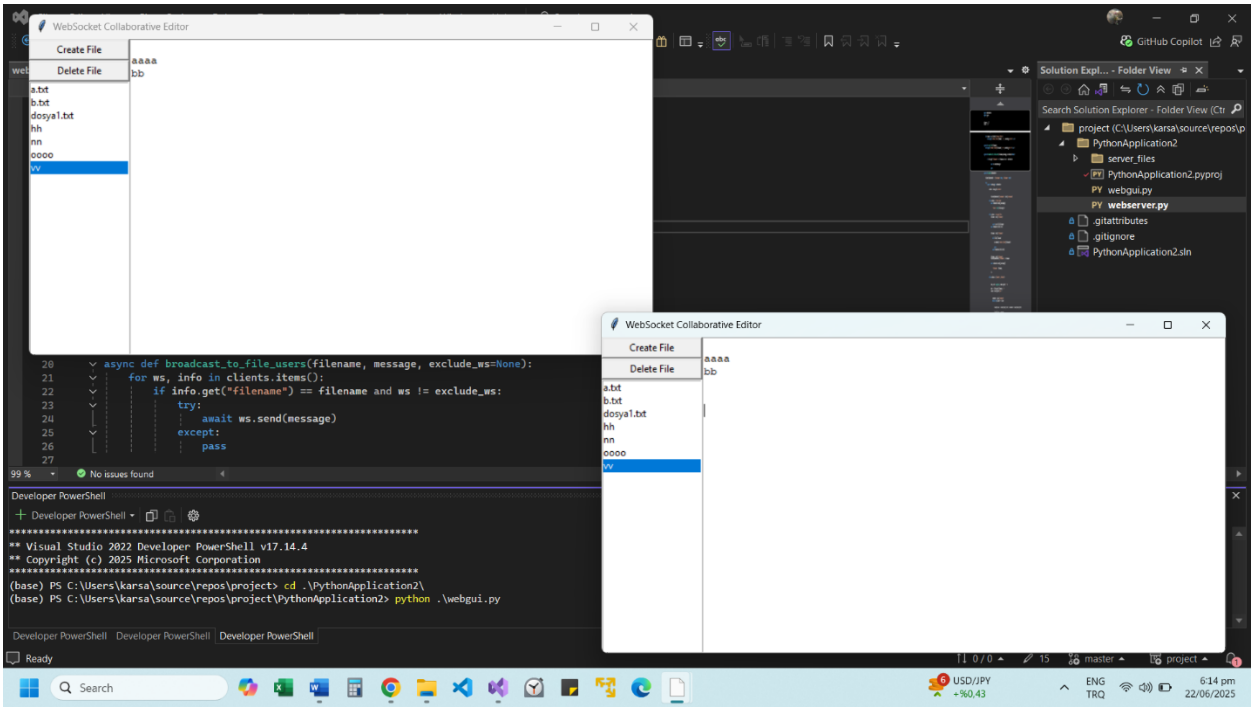
The environment is prepared for multiple users by running the client code again from another terminal.



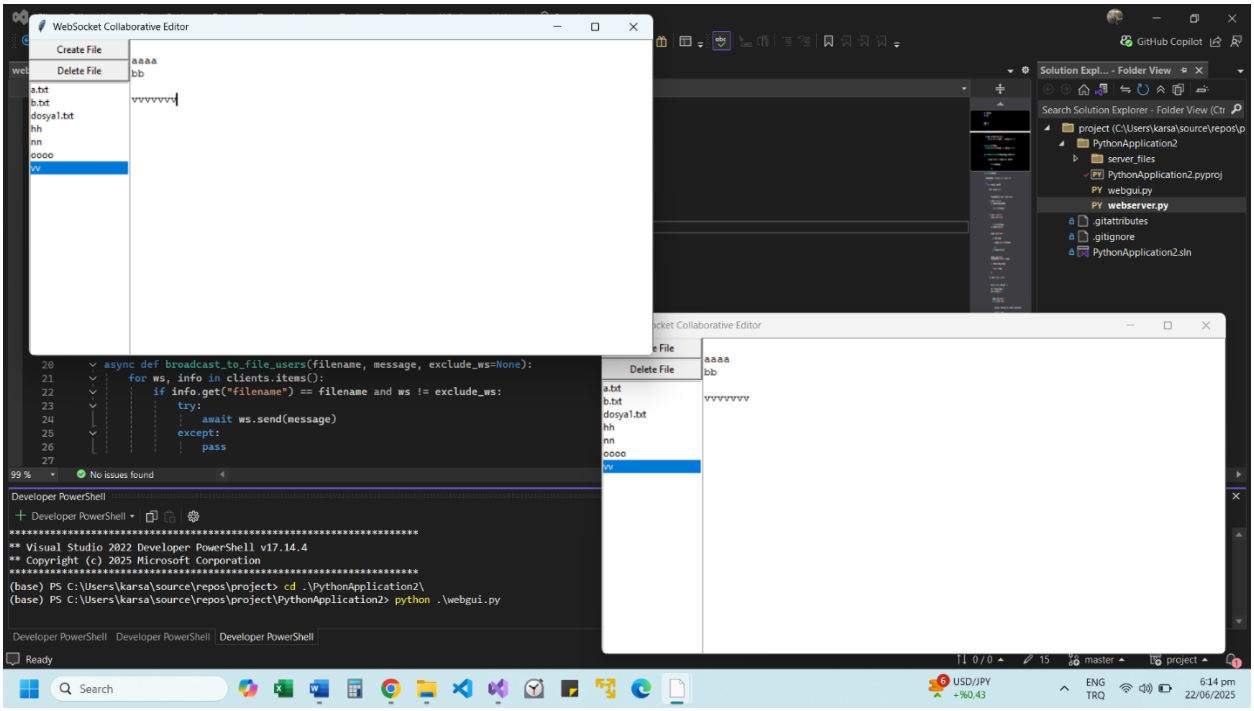
Two users have the text editor opened at the same time. A.txt file content appears.



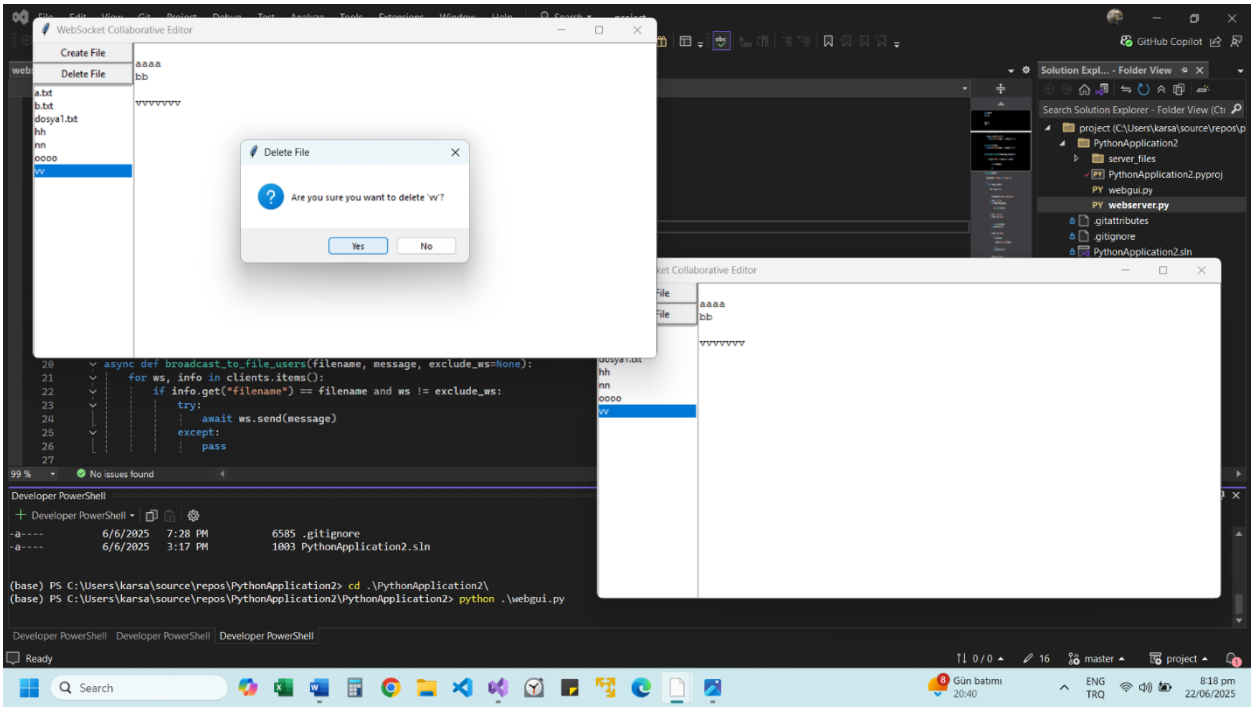
b.txt file content appears.



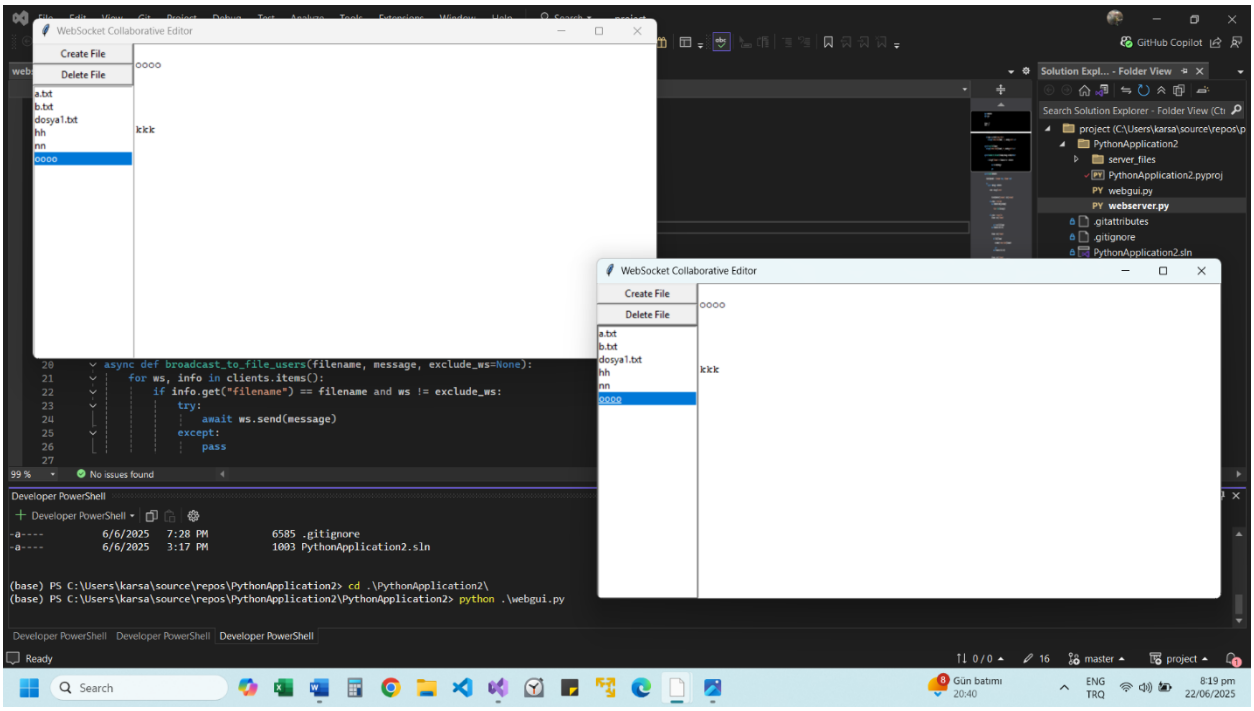
Vv file will be written and tested.



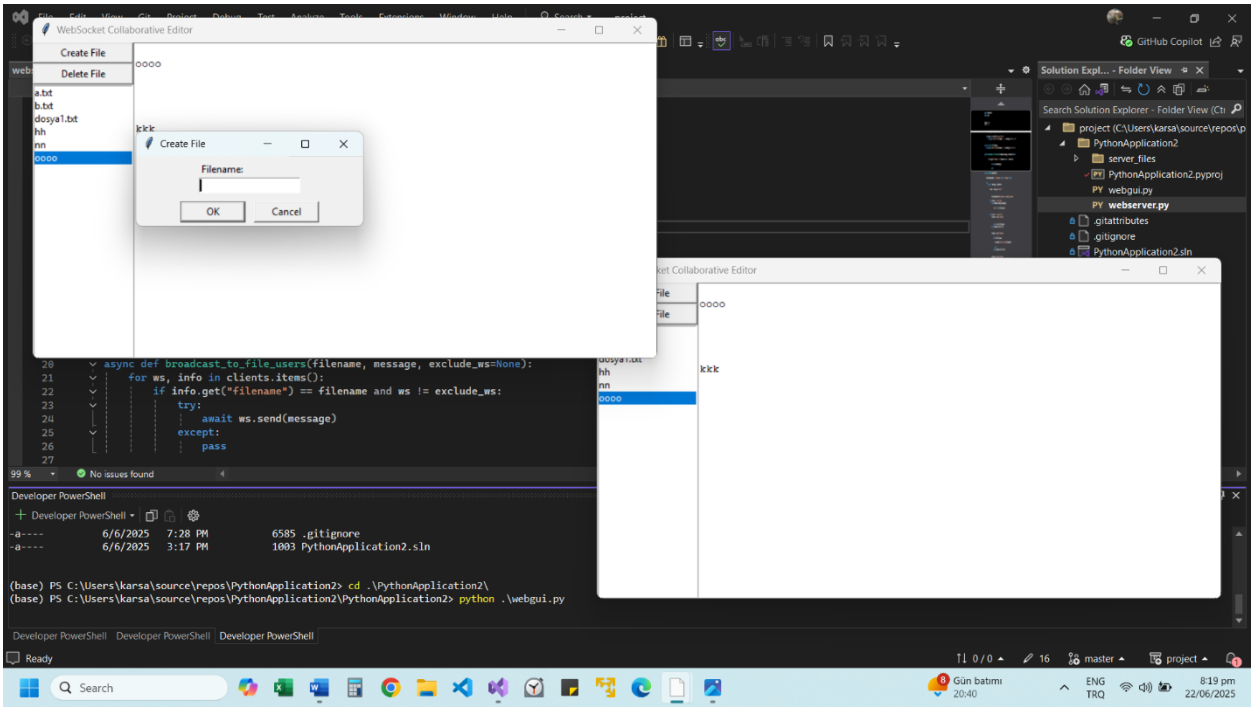
The letter v is written 7 times in the vv file. At the same time, the file content is updated instantly in the other user's interface.



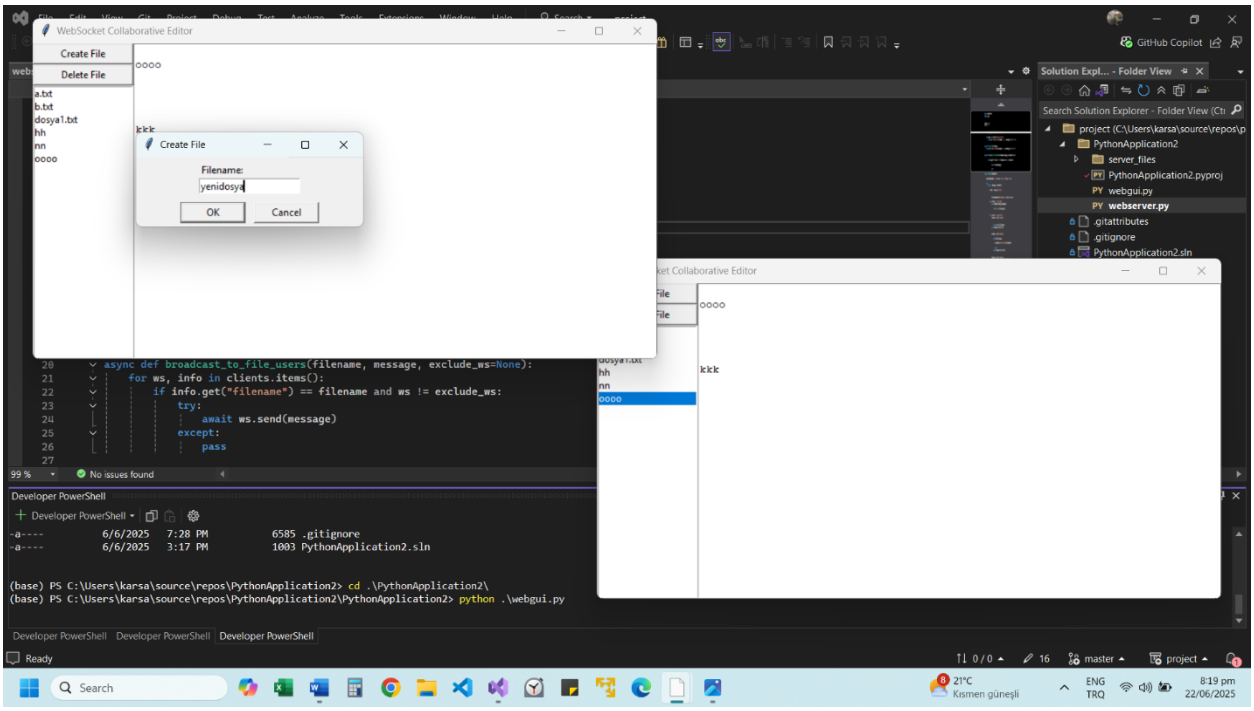
To delete the vv file, click the delete file button.



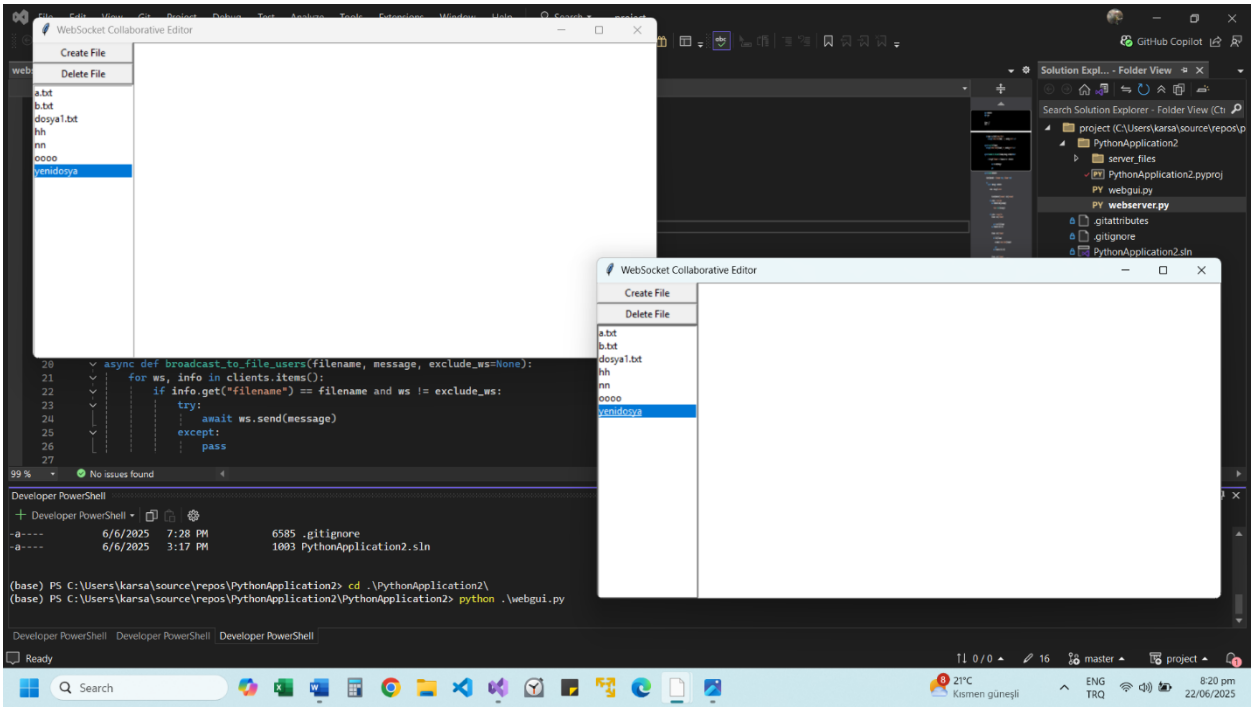
In the screenshot, it is seen that the file was deleted by the other user by deleting the vv file.



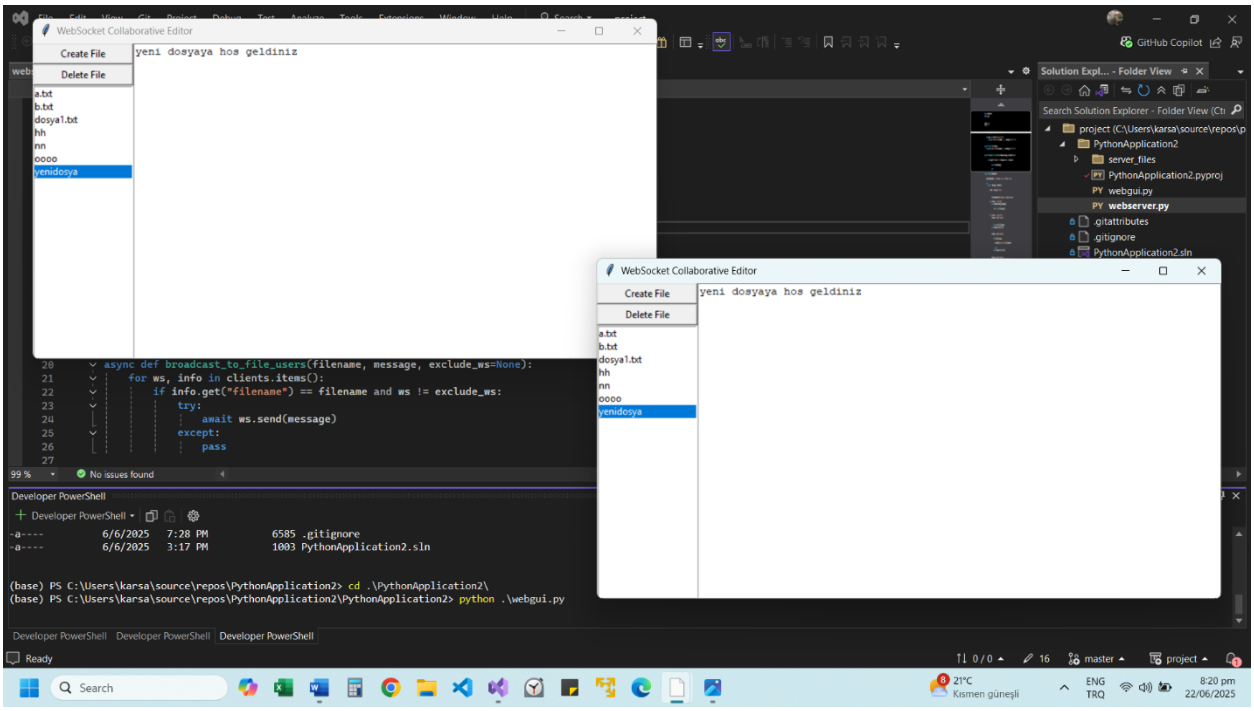
A new file can be created by clicking the Create file button.



It has been named Yenifile.



It can be seen in the screenshot that the file was created.

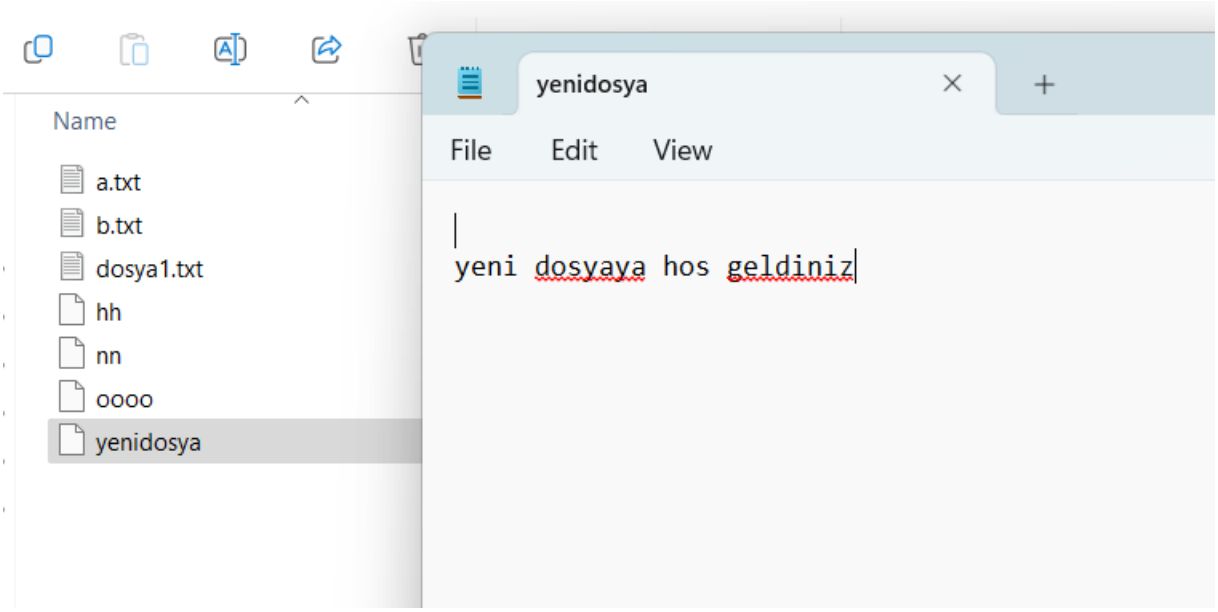


A text has been written in the new file content and it appears instantly in the other file.

server_files	22/06/2025 8:19 pm	File folder	
PythonApplication2.pyproj	21/06/2025 6:33 pm	Python Project	2 KB
webgui.py	22/06/2025 4:16 pm	Python Source File	5 KB
webserver.py	22/06/2025 4:13 pm	Python Source File	5 KB

Files are saved in the server_files folder, as seen below.

a.txt	22/06/2025 6:16 pm	Text Document	1 KB
b.txt	22/06/2025 6:13 pm	Text Document	1 KB
dosya1.txt	21/06/2025 5:42 pm	Text Document	1 KB
hh	21/06/2025 6:31 pm	File	1 KB
nn	21/06/2025 6:37 pm	File	1 KB
oooo	21/06/2025 6:31 pm	File	1 KB
yenidosya	22/06/2025 8:20 pm	File	1 KB



When the content of the new file is examined from the files, the article we wrote was saved.