Computer Networks and Security

University of East London

Faculty of Engineering - Ain Shams University
1839

Phase 2

Omar Sobhy	20P2084
Khaled Hassan	20P9978
Habiba AbdelRazek	20P9595

Presented to Dr. Ayman Bahaa

Stallings 8e

Table of Contents

P2P implemented + Phase 2

Overview

In Phase 2, we enhanced the P2P file sharing system by adding secure user authentication mechanisms. Users must register and log in to perform file sharing actions, ensuring controlled access to shared files.

Implemented Features

- **User Registration and Login**:
  Users can create accounts by choosing a username and password. Authentication is required before accessing any file-sharing operation.
- **Password Hashing**:
  Passwords are not stored in plain text. We use **SHA-256** hashing with a randomly generated salt. User credentials are securely stored in a local JSON file (`user_data/users.json`).
- **Session Management**:
  Upon successful login, users receive a **session token** (32-character hex string).
  Sessions are stored **in-memory** (`active_sessions` dictionary) and have a configurable timeout (default **3 minutes**).
  Expired sessions are automatically invalidated.
- **Authentication-Integrated File Sharing**:
  All file operations (`LIST_FILES`, `DOWNLOAD`, `UPLOAD`) require a valid session token, enforcing that only authenticated users can interact with peers.

Challenges Faced

- **Session Token Management**:
  Implementing secure, per-peer session tokens and ensuring session renewal during activity required careful synchronization between client and server.
- **Handling Peer-to-Peer Dynamic Connections**:
  Each peer maintains independent authentication per connected peer, which added complexity compared to a traditional centralized server model.
- **Partial Data Transfers**:
  During file downloads, ensuring that file transfers were terminated correctly (`<EOF>` marker) without causing connection resets was challenging.

Phase 3 Plan

Overview:

- **File Encryption**
  - Implement AES-256 for file encryption
  - Key derivation from user credentials
- **Integrity Verification**
  - SHA-256 checksums for uploaded files
  - Automatic validation on download
- **Enhanced Security**
  - Upgrade to Argon2 password hashing

1. Upgrade to Argon2 Password Hashing

Current Limitation:

The system uses SHA-256 with salting, which is cryptographically secure but vulnerable to brute-force attacks if the password database is compromised. Modern systems use memory-hard algorithms like Argon2 to resist GPU/ASIC cracking.

Recommendation:

Replace SHA-256 with **Argon2id** (the current industry standard for password hashing). Key advantages:

- **Memory-hard computation**: Requires significant RAM, making large-scale attacks impractical.
- **Time and parallelism tuning**: Can be adjusted to balance security and performance.
- **Built-in salt handling**: Simplifies implementation while maintaining security.

2. Session Persistence

Current Limitation:

Sessions are stored in memory, so they vanish if the peer server restarts, forcing users to reauthenticate.

Recommendation:

Persist sessions to a **lightweight database** (e.g., SQLite) with:

- **Encrypted session tokens**: Store only hashed tokens to prevent misuse if the database is breached.
- **Automatic cleanup**: Regularly purge expired sessions.

Implementation Plan

1. File Encryption on Upload

Approach:

- Use **AES-256-GCM** (symmetric encryption) for files:
  - Combines encryption *and* authentication (detects tampering).
  - Operates in "streaming mode" to handle large files efficiently.
- Generate a **unique symmetric key per file** (not shared globally).

Workflow:

1. User selects a file to upload.
2. System:
   - Generates a random AES key (`secrets.token_bytes(32)`).
   - Encrypts the file in chunks (to avoid memory overload).
   - Stores the encrypted file in `shared_files/`.

2. File Decryption on Download

Approach:

- Only users with the correct key can decrypt files.
- Decryption happens *after* file transfer to avoid exposing plaintext.

Workflow:

1. User requests a file download.
2. System:
   - Transfers the encrypted file (as binary chunks).
   - Provides the AES key to the recipient (via secure channel).
   - Recipient decrypts the file locally using the key.

3. File Integrity Verification

Approach:

- Use **SHA-256 hashes** to verify file integrity:
  - Generate a hash *before encryption* (stores with file metadata).
  - Recompute hash *after decryption* and compare.

Workflow:

1. On upload:
   - Compute SHA-256(file_content) → store as file_hash.
2. On download:
   - Recompute SHA-256(decrypted_content).
   - Abort if hashes mismatch (tampering detected).

4. Key Management

Simplified Approach:

- **Option A (Single Key)**:
  - Use *one* pre-shared AES key for *all* files (easiest but insecure).
  - **Risk**: Compromised key exposes all files.
- **Option B (Per-File Keys)**:
  - Generate a unique AES key per file.
  - Share keys securely:
    - Encrypt the AES key with the *recipient's public key* (if PKI is available).
    - **Fallback**: Use **password-derived keys** (e.g., hash the recipient's password + salt).
- **Option C (Diffie-Hellman Lite)**:
  - Peers agree on a shared secret during session setup.
  - Derive file keys from this secret.
  - **Trade-off**: More complex but eliminates key storage.

Upload Flow	Transfer Flow	Download Flow
• Hash file (SHA-256) • 2. Encrypt (AES-256)	• Send encrypted chunks over TCP • Send metadata: ○ File hash ○ - Key (secured)	• Receive encrypted chunks • Decrypt locally with provided key • 3. Verify hash

Network Architecture Overview

The system consists of two major components:

Component	Role
Rendezvous Server (rendezvous_server.py)	Simple centralized server used ONLY for helping peers discover each other.

Component	Role
Peer Node (peer.py)	Each peer is both a server (serving files) and a client (requesting files).

2. Peer Behavior

Every `peer.py` node does **two things simultaneously**:

- **Server Mode**:
  - Listens for incoming connections from other peers.
  - Serves file requests (list, download, upload) **only to authenticated users**.
- **Client Mode**:
  - Connects to other peers to request files or upload files.
  - Handles login/registration to each peer separately.

3. Discovery Mechanism: Rendezvous Server

- When a peer starts:
  1. It connects to the **Rendezvous Server** (`rendezvous_server.py`).
  2. Sends a `REGISTER <IP> <PORT>` command to announce itself.
  3. The server adds it to the known peer list.
  4. The server sends back a list of **other currently active peers** (except the newly joined one).
- **Peers can refresh the peer list dynamically** during runtime by connecting again to the rendezvous server.

4. Communication and Operations

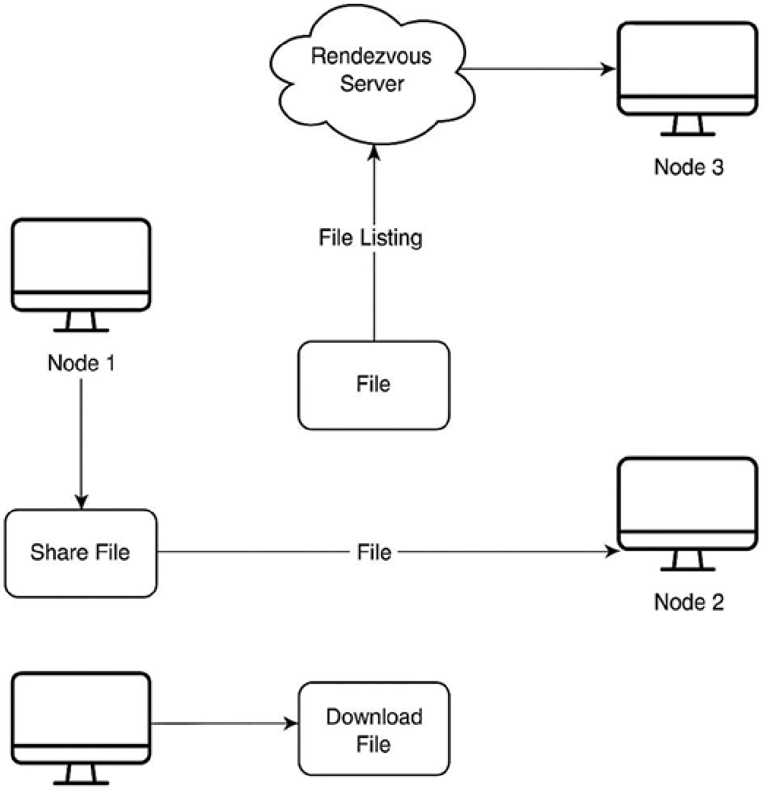
After discovery, a peer can:

Operation	How It Works
Login/Register	A peer must login/register to another peer to get a session token.
List Files	Authenticated peer sends session_token LIST_FILES and receives file names.
Download File	Authenticated peer sends session_token DOWNLOAD filename and receives file content.
Upload File	Authenticated peer sends session_token UPLOAD filename, sends file, and gets confirmation.

5. Authentication and Session Management

- **Usernames and passwords** are created during registration and are hashed with **SHA-256** and a random salt.
- Upon successful login, a **unique session token** is issued.
- Session tokens are stored **in-memory** and expire after inactivity (3 minutes).

System Architecture



Authentication Design Document

1. User Credentials

- Usernames and passwords are collected at registration.
- Passwords are **hashed using SHA-256** combined with a randomly generated **salt**.
- Stored in `user_data/users.json` in the format:

```
{
  "username1": {
    "password_hash": "hashed_password_here",
    "salt": "random_salt_here"
  },
  "username2": { ... }
}
```

2. Password Hashing Algorithm

- **Algorithm Used**: SHA-256
- **Salt Generation**: `secrets.token_hex(16)` (16 random hex characters)
- **Hashing Process**:

```
hashed_password = SHA-256(password + salt)
```

```
def hash_password(password, salt=None):
    if not salt:
        salt = secrets.token_hex(16)
    salted = password + salt
    return hashlib.sha256(salted.encode()).hexdigest(), salt
```

- **Security Properties**:
  - Salt prevents rainbow table attacks
  - SHA-256 provides collision resistance
  - Future upgrade path to Argon2

3. Session Management

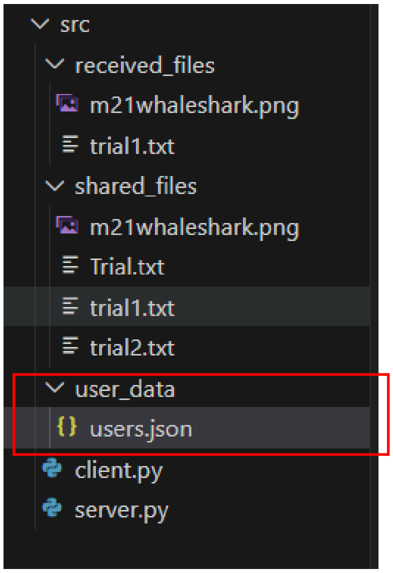
- On successful login, a **session token** (32 hex characters) is issued.
- Session data is stored in-memory inside:

```
active_sessions = {
    session_token: {
        "username": username,
        "expiry": datetime.now() + timedelta(minutes=3)
    }
}
```

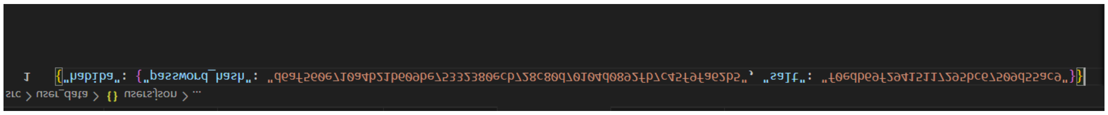
- **Session Expiry**: Sessions expire after 30 minutes of inactivity. Renewal happens automatically on each valid operation.
- **Validation**:
  - Incoming commands must start with a valid session token.
  - If session is missing or expired, peer responds with authentication error.

Password handling:

```
def hash_password(password, salt=None):
    if salt is None:
        salt = secrets.token_hex(16)
    hashed = hashlib.sha256((password + salt).encode()).hexdigest()
    return hashed, salt

def verify_password(stored_hash, salt, input_password):
    input_hash = hashlib.sha256((input_password + salt).encode()).hexdigest()
    return input_hash == stored_hash

USER_DATA_FILE = "user_data/users.json"

def load_user_data():
    if not os.path.exists(USER_DATA_FILE):
        os.makedirs(os.path.dirname(USER_DATA_FILE), exist_ok=True)
        with open(USER_DATA_FILE, 'w') as f:
            json.dump({}, f)
        return {}
    with open(USER_DATA_FILE, 'r') as f:
        return json.load(f)

def save_user_data(data):
    with open(USER_DATA_FILE, 'w') as f:
        json.dump(data, f)
```

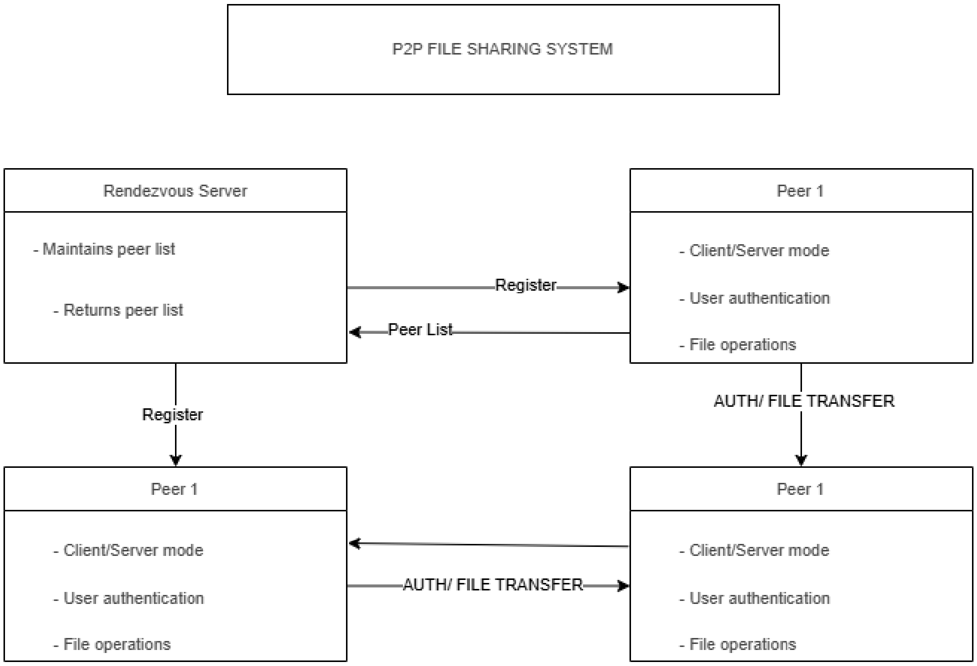


```
src
  received_files
    m21whaleshark.png
    trial1.txt
  shared_files
    m21whaleshark.png
    Trial.txt
    trial1.txt
    trial2.txt
  user_data
    users.json
  client.py
  server.py
```



```
1    {"habiba": {"password_hash": "[illegible]", "salt": "[illegible]"}}
src > user_data > {} users.json > ...
```

System Architecture



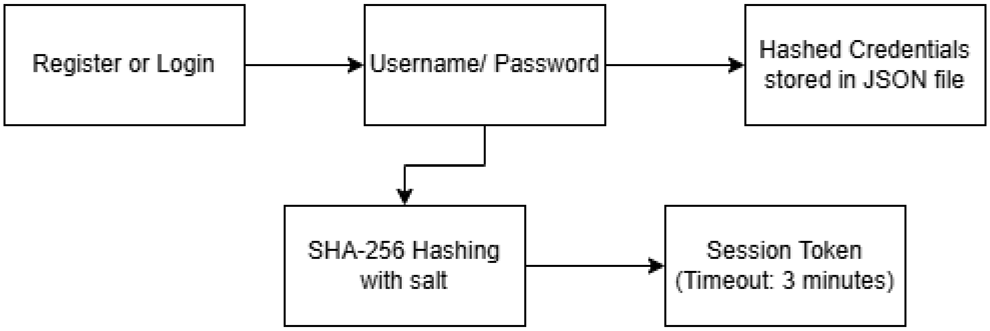
COMPONENT DETAILS:

1. Rendezvous Server:
   - Centralized peer discovery
   - Maintains list of active peers
   - Returns peer list to registering nodes
   - Simple TCP server (port 5000)
2. Peer Nodes (Each peer has both client and server components):

Peer Server	Peer Client
• Listens on TCP port • Handles: ○ Authentication ○ File Operation • Session Management • User Data Storage	• Connects to other peers • Provides user menu • Initiates: ○ Authentication ○ File Transfers

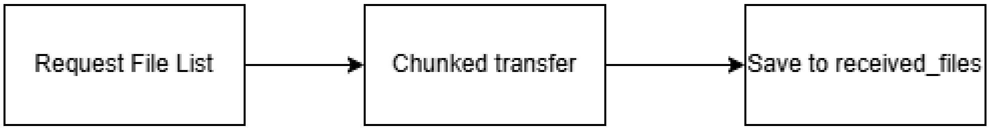
AUTHENTICATION FLOW:

1. Peer registers with Rendezvous Server
2. Peer gets list of available peers
3. User selects peer and chooses:

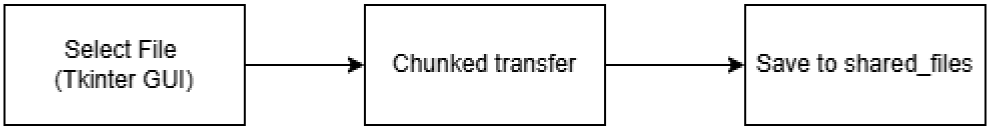

Register or Login
Username/ Password
Hashed Credentials stored in JSON file
SHA-256 Hashing with salt
Session Token (Timeout: 3 minutes)

FILE TRANSFER FLOW (Authenticated):

1. User selects operation (upload/download)
2. Client verifies session token
3. **For download:**



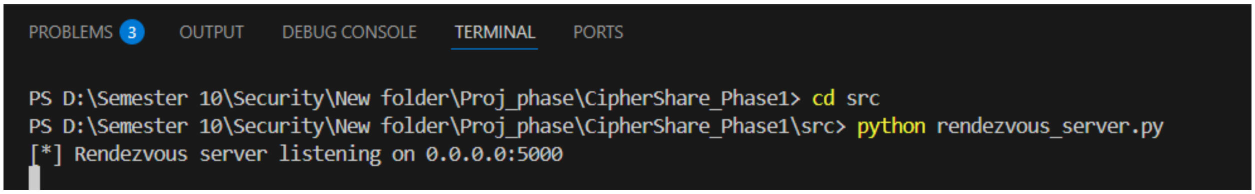
1. **For upload:**



User Manual

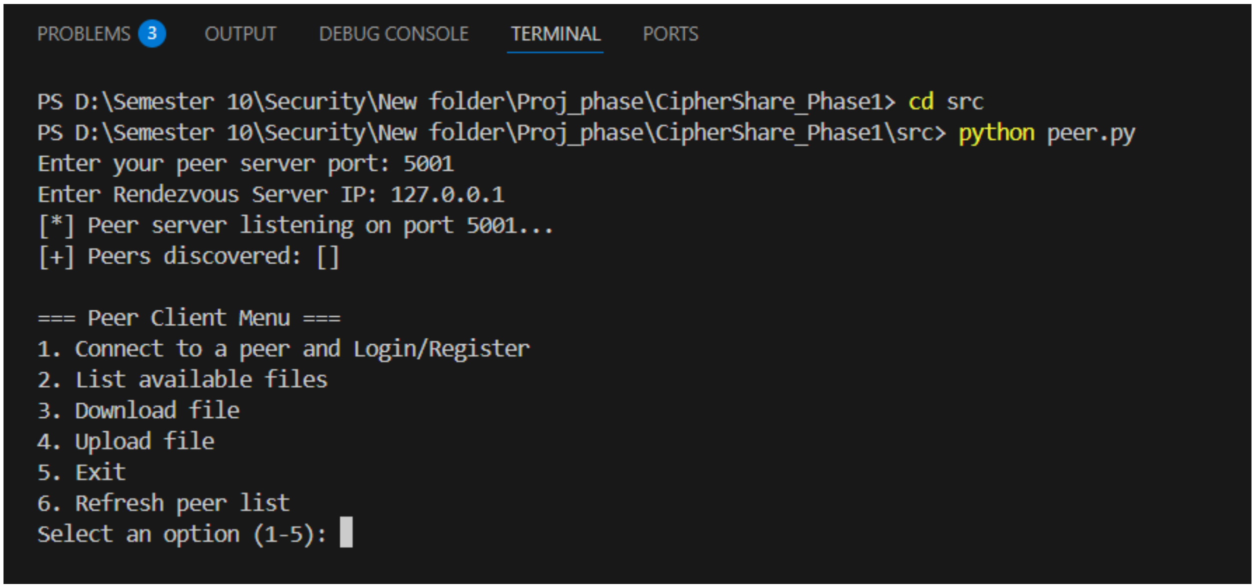
Step 1:

- **Run rendezvous_server.py**
- **Run 2 peer.py**



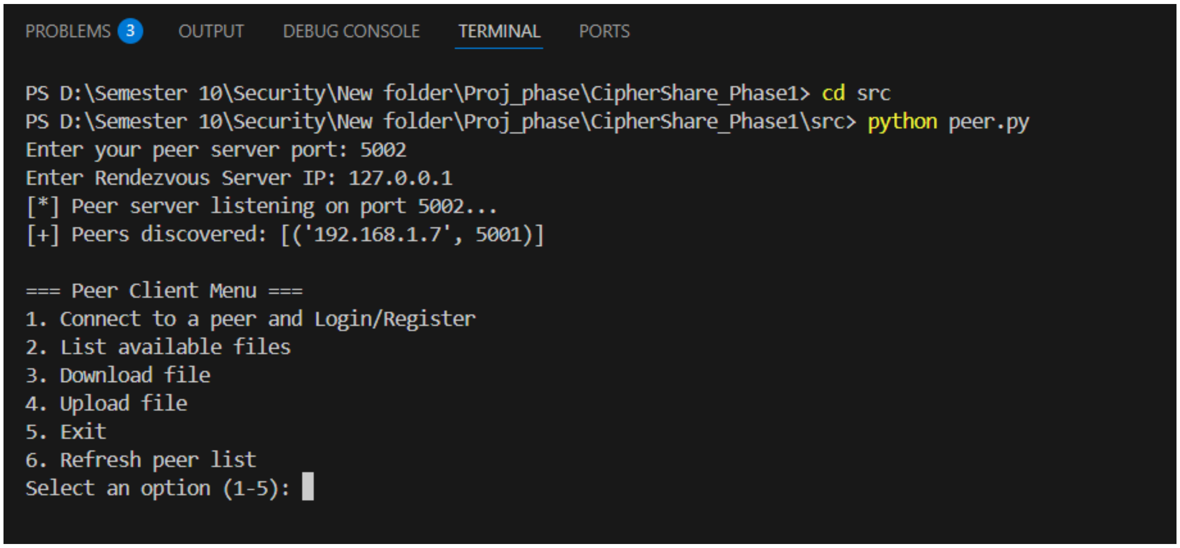
```
PS D:\Semester 10\Security\New folder\Proj_phase\CipherShare_Phase1> cd src
PS D:\Semester 10\Security\New folder\Proj_phase\CipherShare_Phase1\src> python rendezvous_server.py
[*] Rendezvous server listening on 0.0.0.0:5000
```

Peer 1:



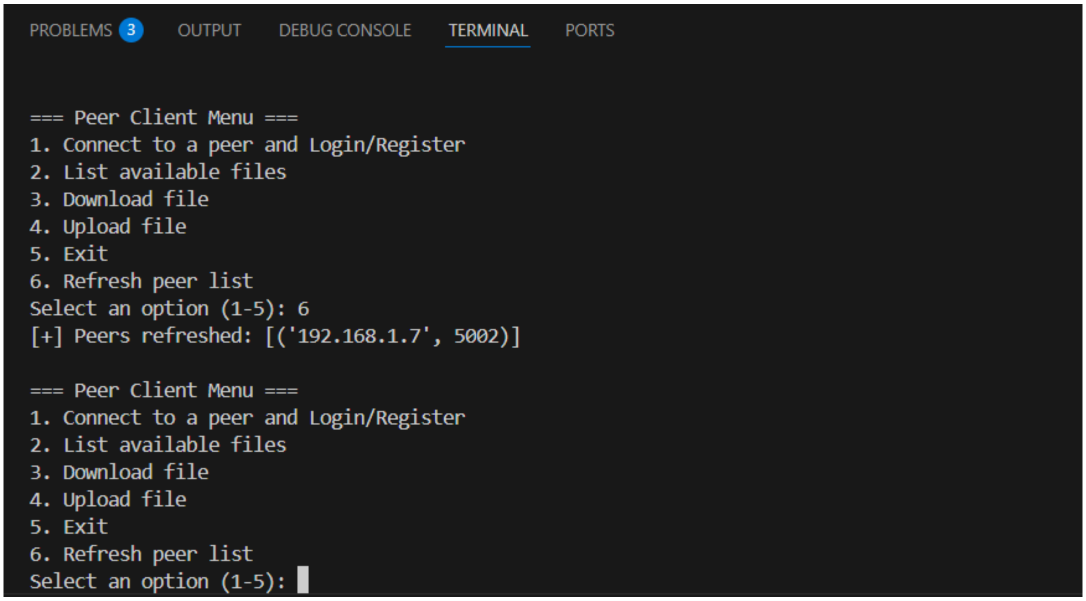
```
PS D:\Semester 10\Security\New folder\Proj_phase\CipherShare_Phase1> cd src
PS D:\Semester 10\Security\New folder\Proj_phase\CipherShare_Phase1\src> python peer.py
Enter your peer server port: 5001
Enter Rendezvous Server IP: 127.0.0.1
[*] Peer server listening on port 5001...
[+] Peers discovered: []

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5):
```

Peer 2:



```
PS D:\Semester 10\Security\New folder\Proj_phase\CipherShare_Phase1> cd src
PS D:\Semester 10\Security\New folder\Proj_phase\CipherShare_Phase1\src> python peer.py
Enter your peer server port: 5002
Enter Rendezvous Server IP: 127.0.0.1
[*] Peer server listening on port 5002...
[+] Peers discovered: [('192.168.1.7', 5001)]

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5):
```

Refresh Peer list in Peer 1 terminal:

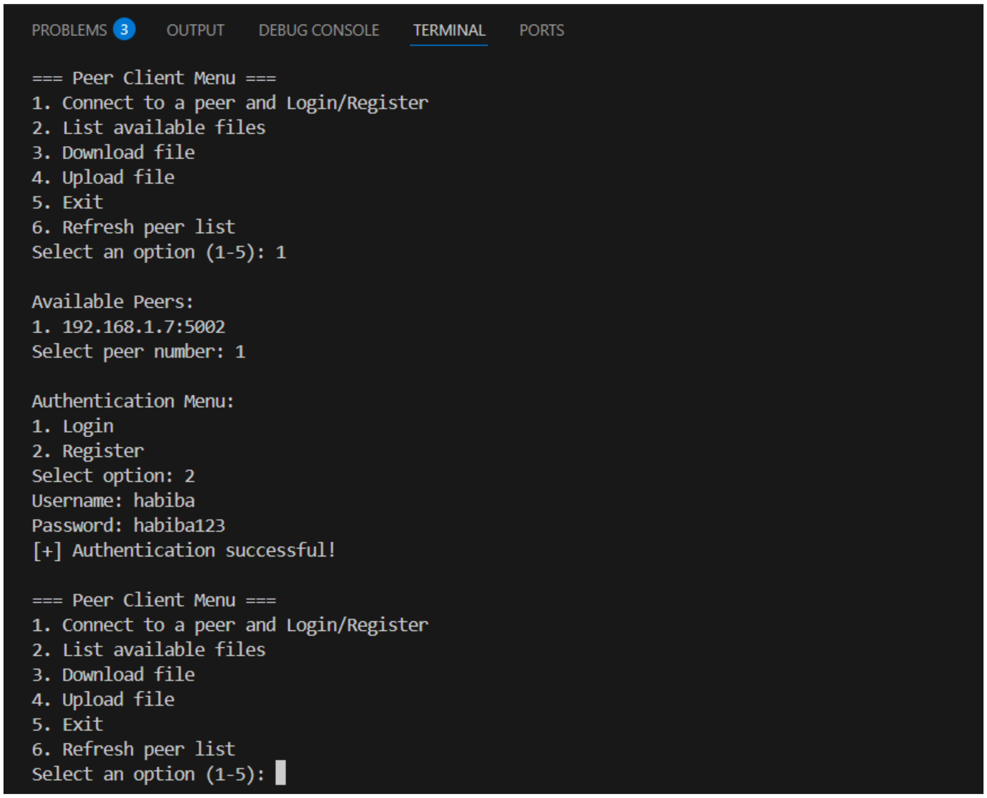


```
=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 6
[+] Peers refreshed: [('192.168.1.7', 5002)]

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 
```

Step 2:

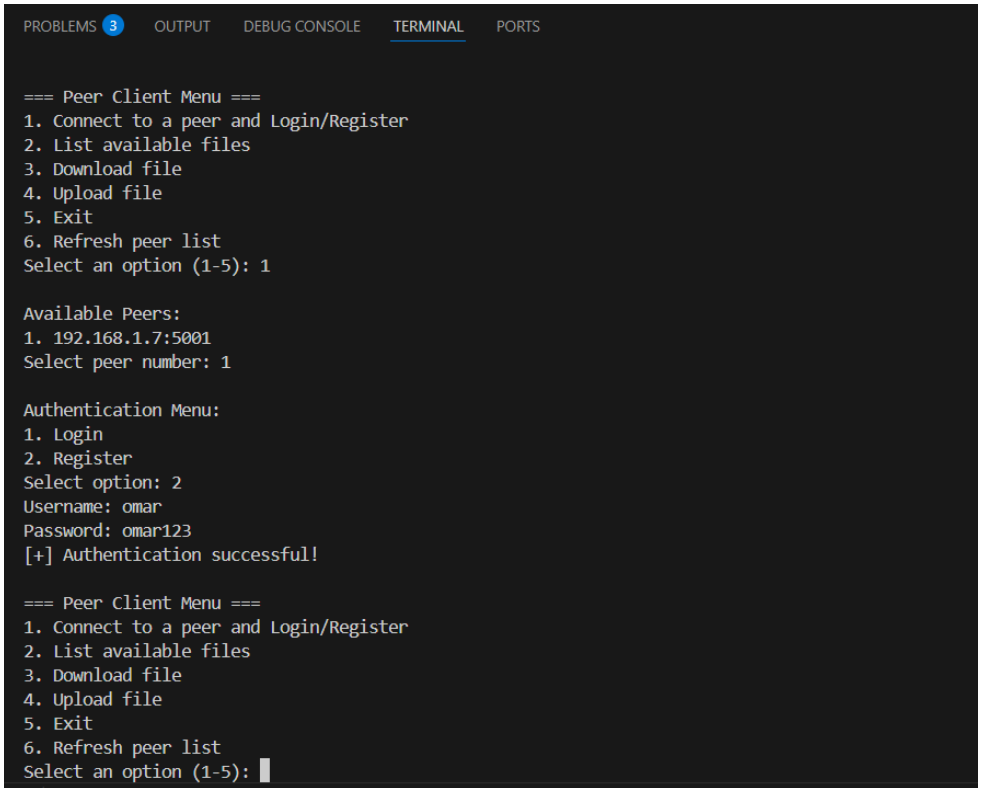
Connect and Register peers

Peer 1:



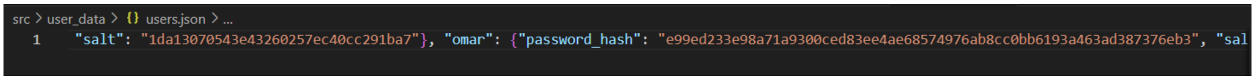
```
=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 1

Available Peers:
1. 192.168.1.7:5002
Select peer number: 1

Authentication Menu:
1. Login
2. Register
Select option: 2
Username: habiba
Password: habiba123
[+] Authentication successful!

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 
```

Peer 2:



```
=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 1

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1

Authentication Menu:
1. Login
2. Register
Select option: 2
Username: omar
Password: omar123
[+] Authentication successful!

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5):
```

Users.json:

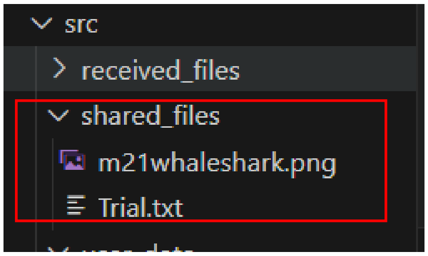


```
"salt": "1da13070543e43260257ec40cc291ba7"}, "omar": {"password_hash": "e99ed233e98a71a9300ced83ee4ae68574976ab8cc0bb6193a463ad387376eb3", "sal
```

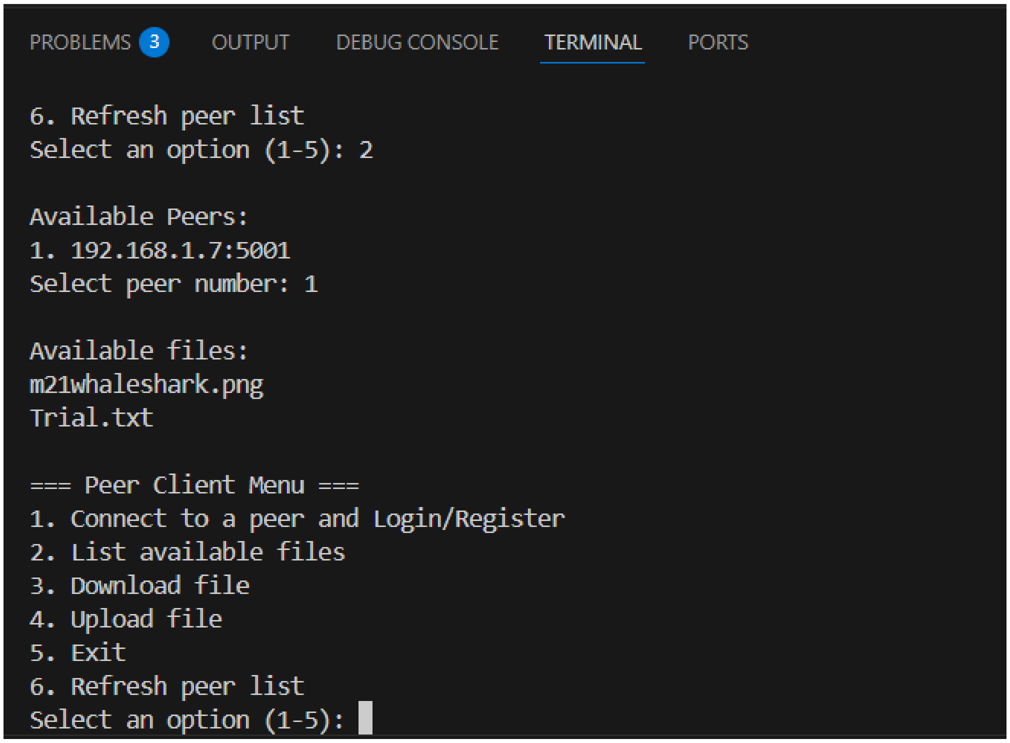
Step 3:

Show List of files:

It displays the list from shared_files folder



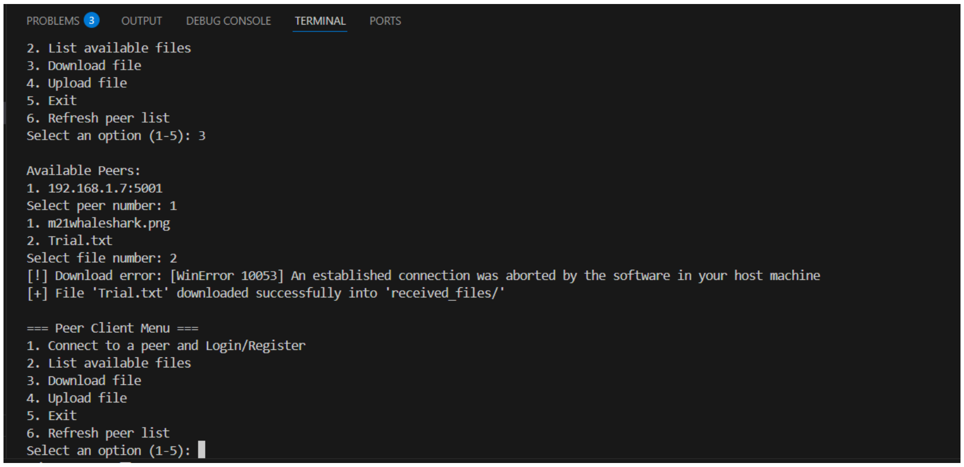
```
src
  received_files
  shared_files
    m21whaleshark.png
    Trial.txt
```



```
6. Refresh peer list
Select an option (1-5): 2

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1

Available files:
m21whaleshark.png
Trial.txt

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5):
```

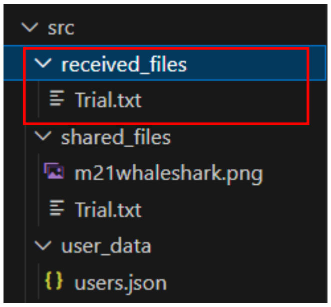
Download file

It downloads from the list of files in the shared_files



```
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 3

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1
1. m21whaleshark.png
2. Trial.txt
Select file number: 2
[!] Download error: [WinError 10053] An established connection was aborted by the software in your host machine
[+] File 'Trial.txt' downloaded successfully into 'received_files/'

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5):
```

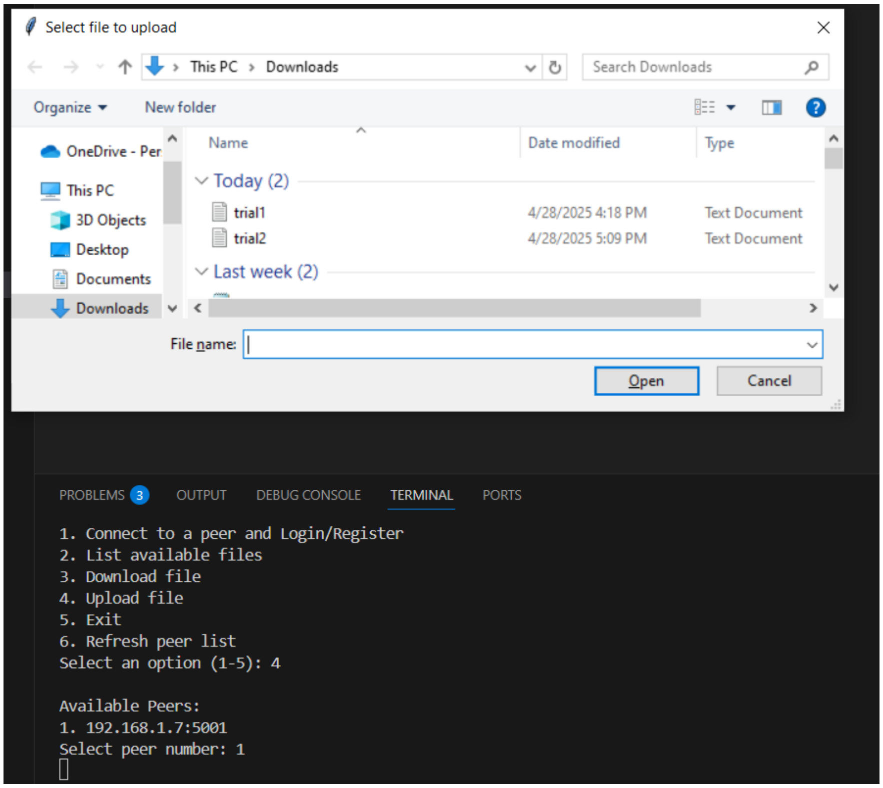
It is saved in folder received_files



```
src
  received_files
    Trial.txt
  shared_files
    m21whaleshark.png
    Trial.txt
  user_data
    users.json
```

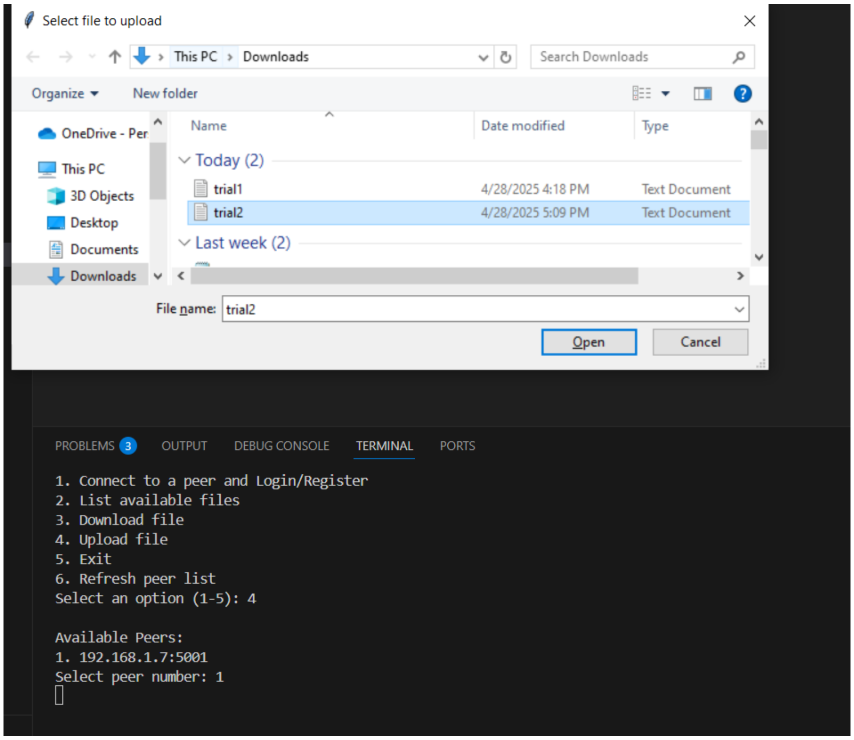
Upload file:

It opens the file explorer window to select file you want to share with other peers



```
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 4

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1
```

Select the file and click 'open'

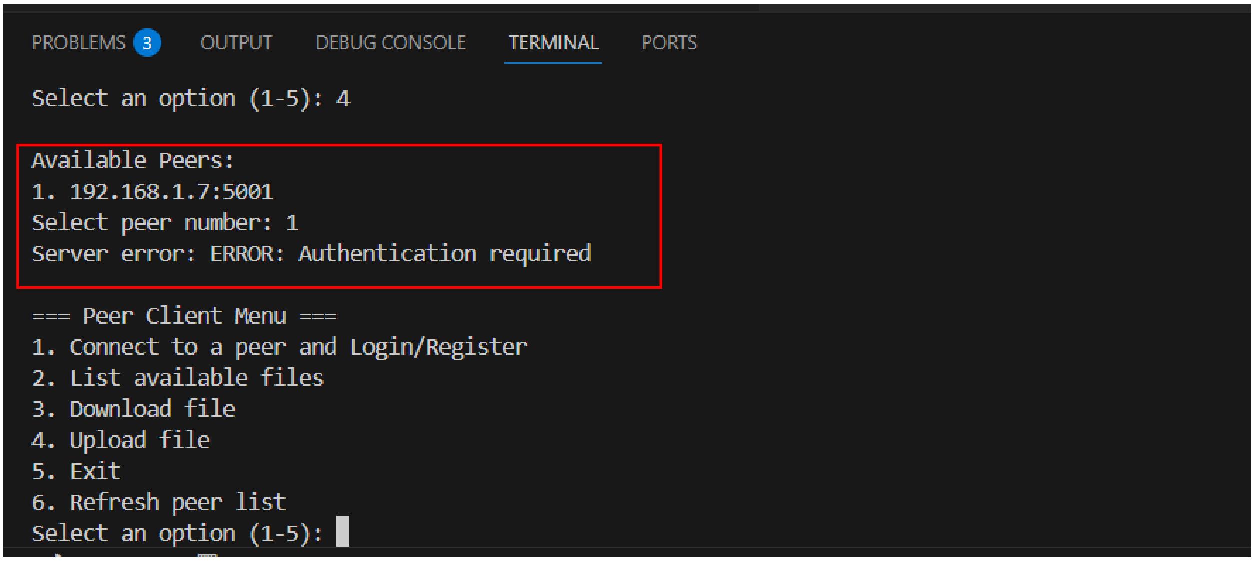


Name	Date modified	Type
trial1	4/28/2025 4:18 PM	Text Document
trial2	4/28/2025 5:09 PM	Text Document

File name: trial2

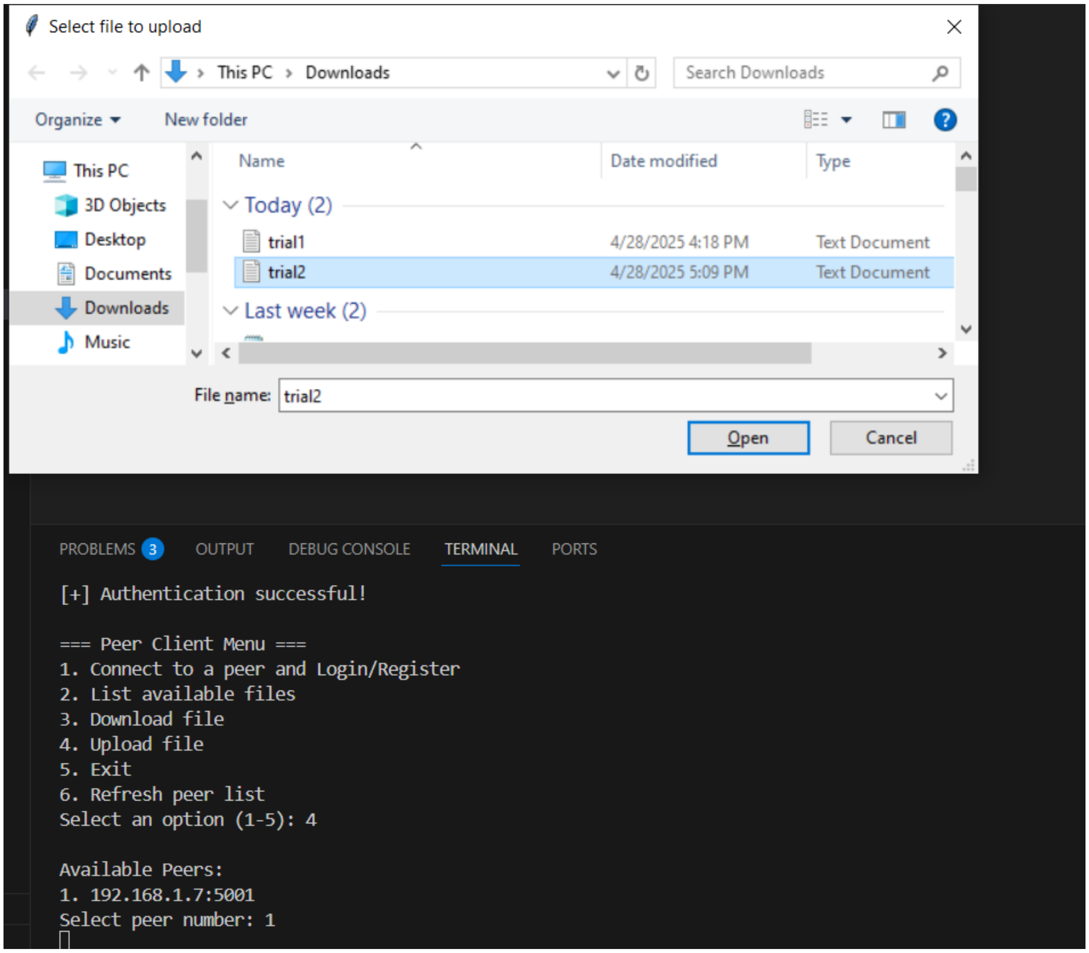
```
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 4

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1
```

After selecting, the session was opened for a while so it timed out and shows that authentication again is needed:



```
Select an option (1-5): 4

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1
Server error: ERROR: Authentication required

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5):
```

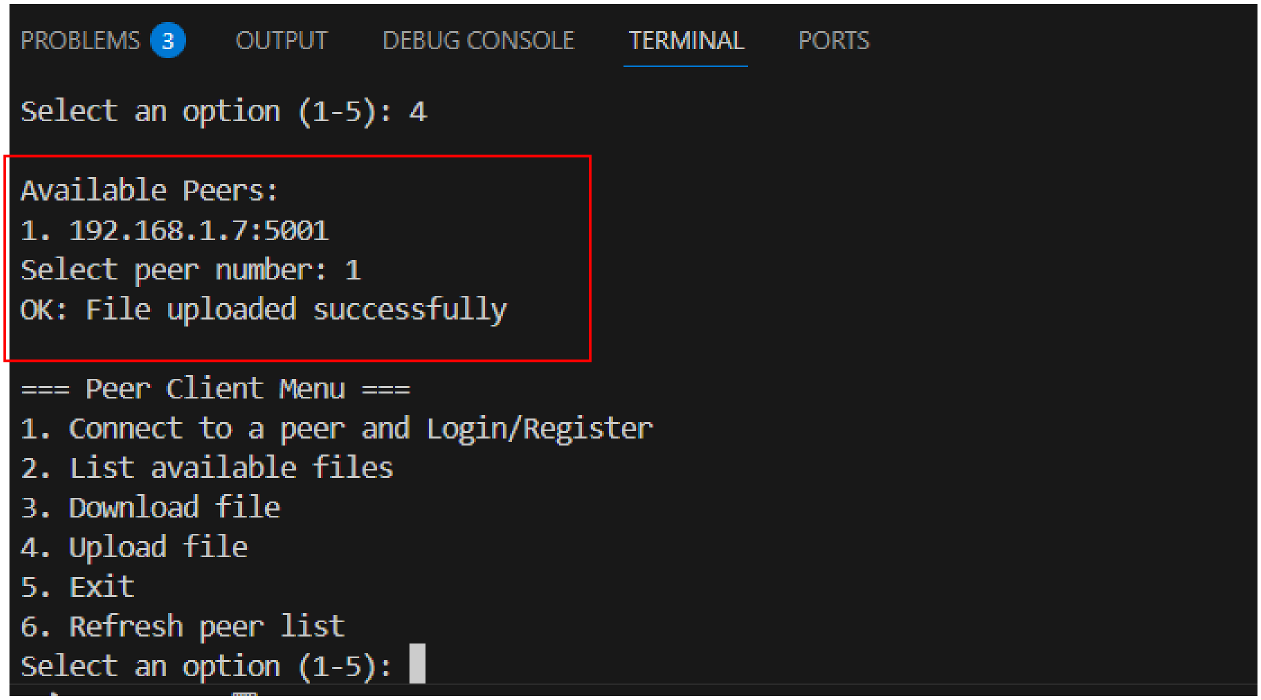
Continue after logging in again:



Name	Date modified	Type
trial1	4/28/2025 4:18 PM	Text Document
trial2	4/28/2025 5:09 PM	Text Document

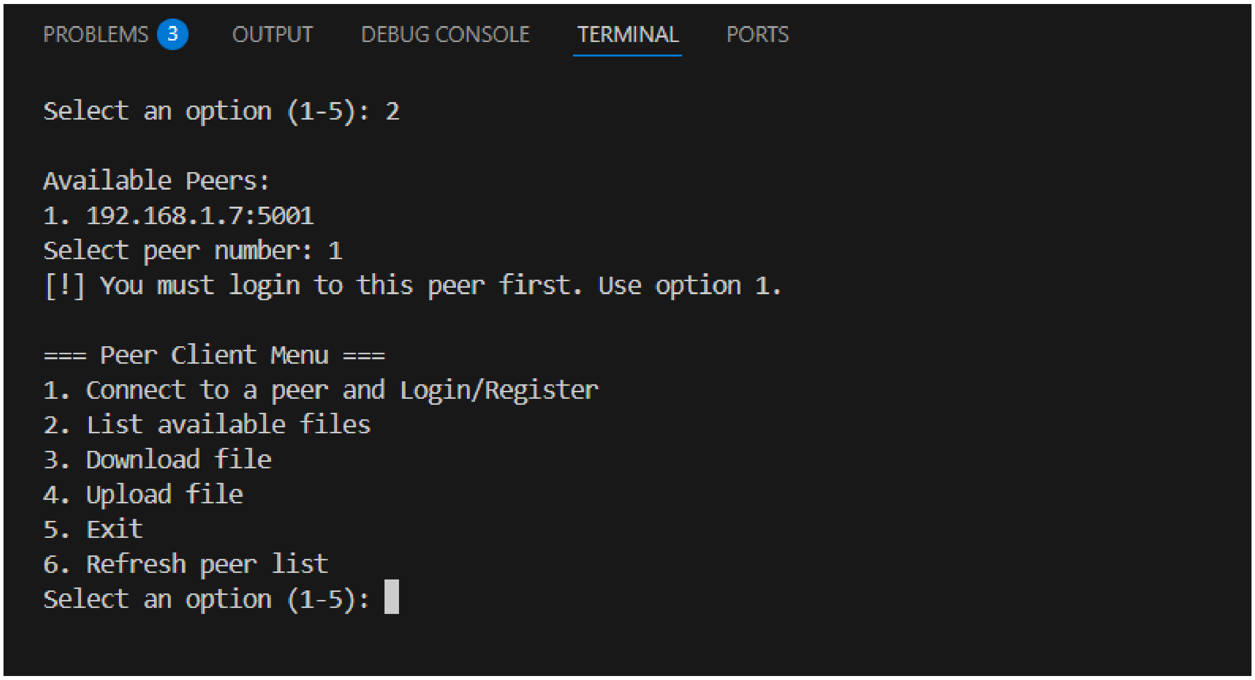
File name: trial2

```
[+] Authentication successful!

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 4

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1
```



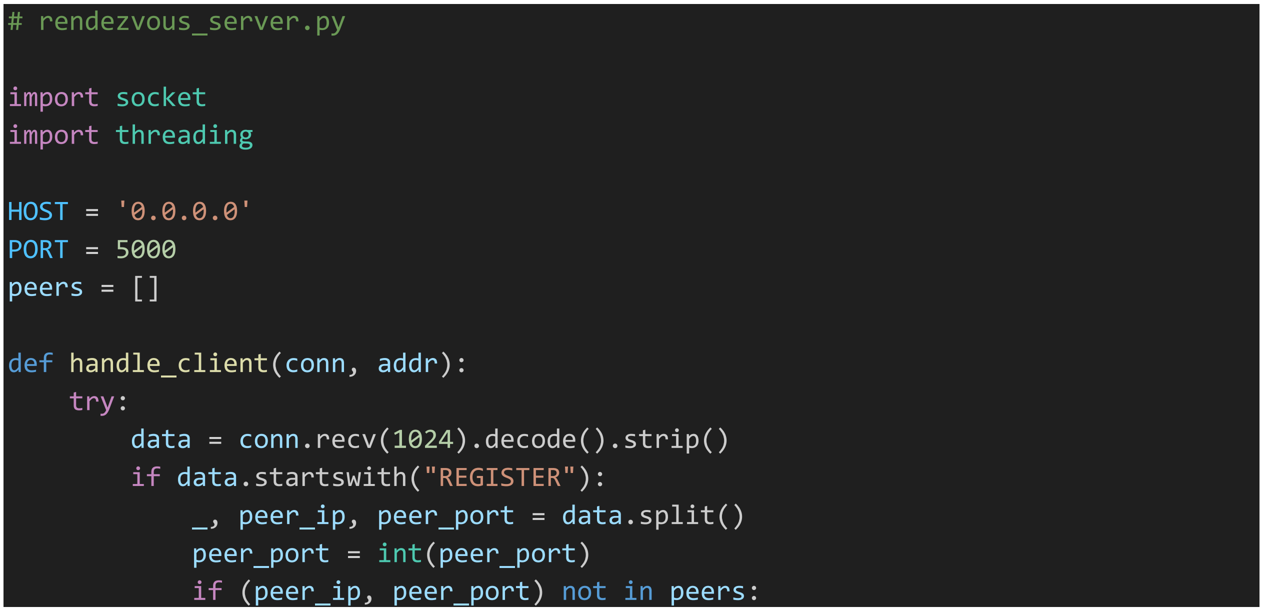
```
Select an option (1-5): 4

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1
OK: File uploaded successfully

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5):
```

If not logged:

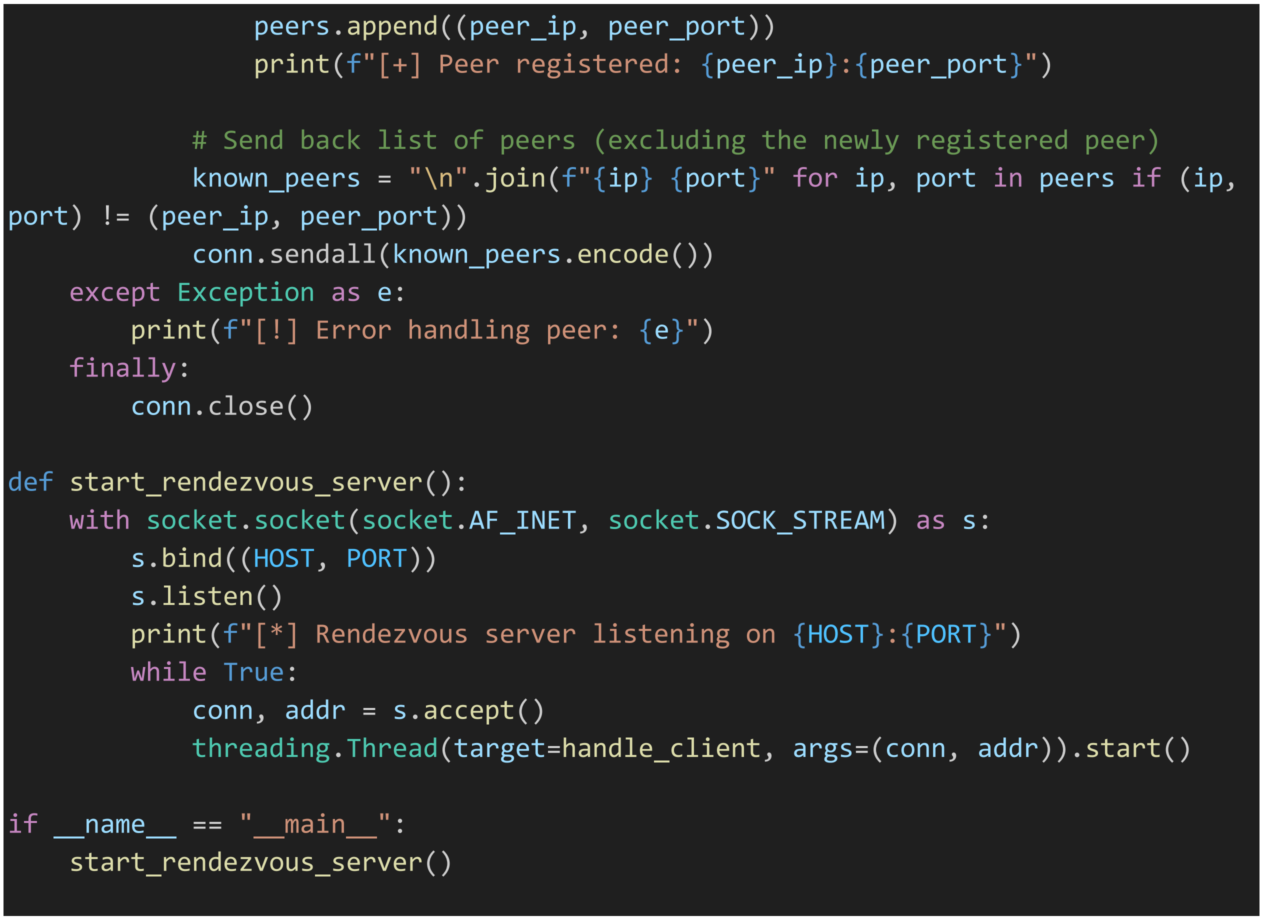


```
Select an option (1-5): 2

Available Peers:
1. 192.168.1.7:5001
Select peer number: 1
[!] You must login to this peer first. Use option 1.

=== Peer Client Menu ===
1. Connect to a peer and Login/Register
2. List available files
3. Download file
4. Upload file
5. Exit
6. Refresh peer list
Select an option (1-5): 
```

File rendezvous_server.py:

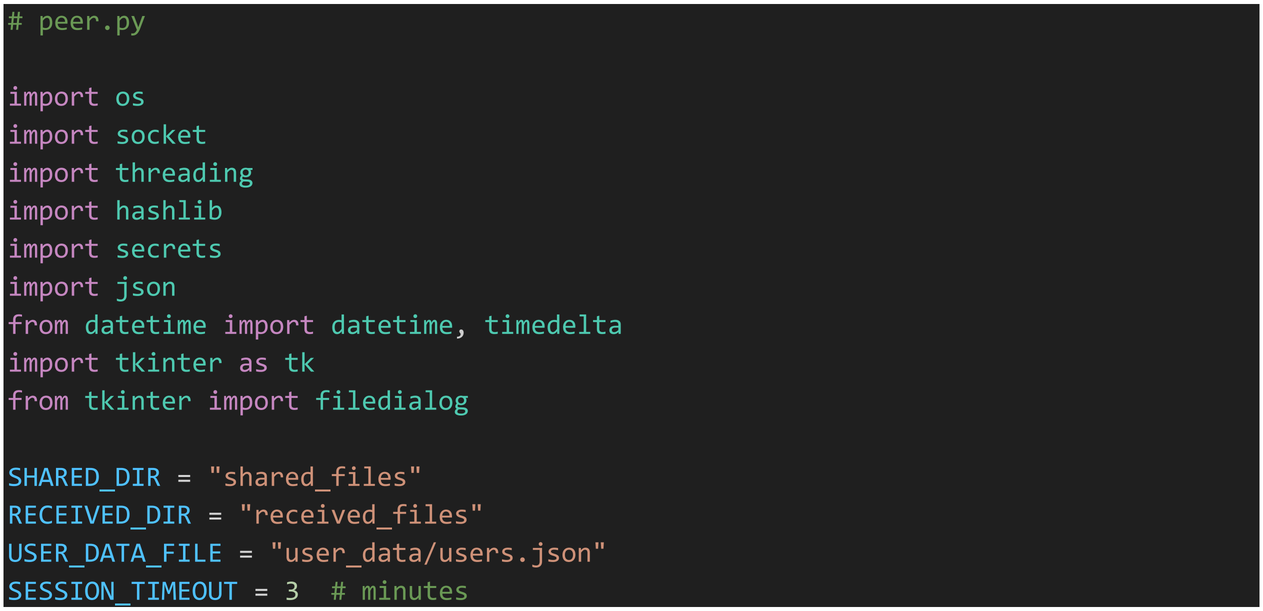


```python
# rendezvous_server.py

import socket
import threading

HOST = '0.0.0.0'
PORT = 5000
peers = []

def handle_client(conn, addr):
    try:
        data = conn.recv(1024).decode().strip()
        if data.startswith("REGISTER"):
            _, peer_ip, peer_port = data.split()
            peer_port = int(peer_port)
            if (peer_ip, peer_port) not in peers:
```

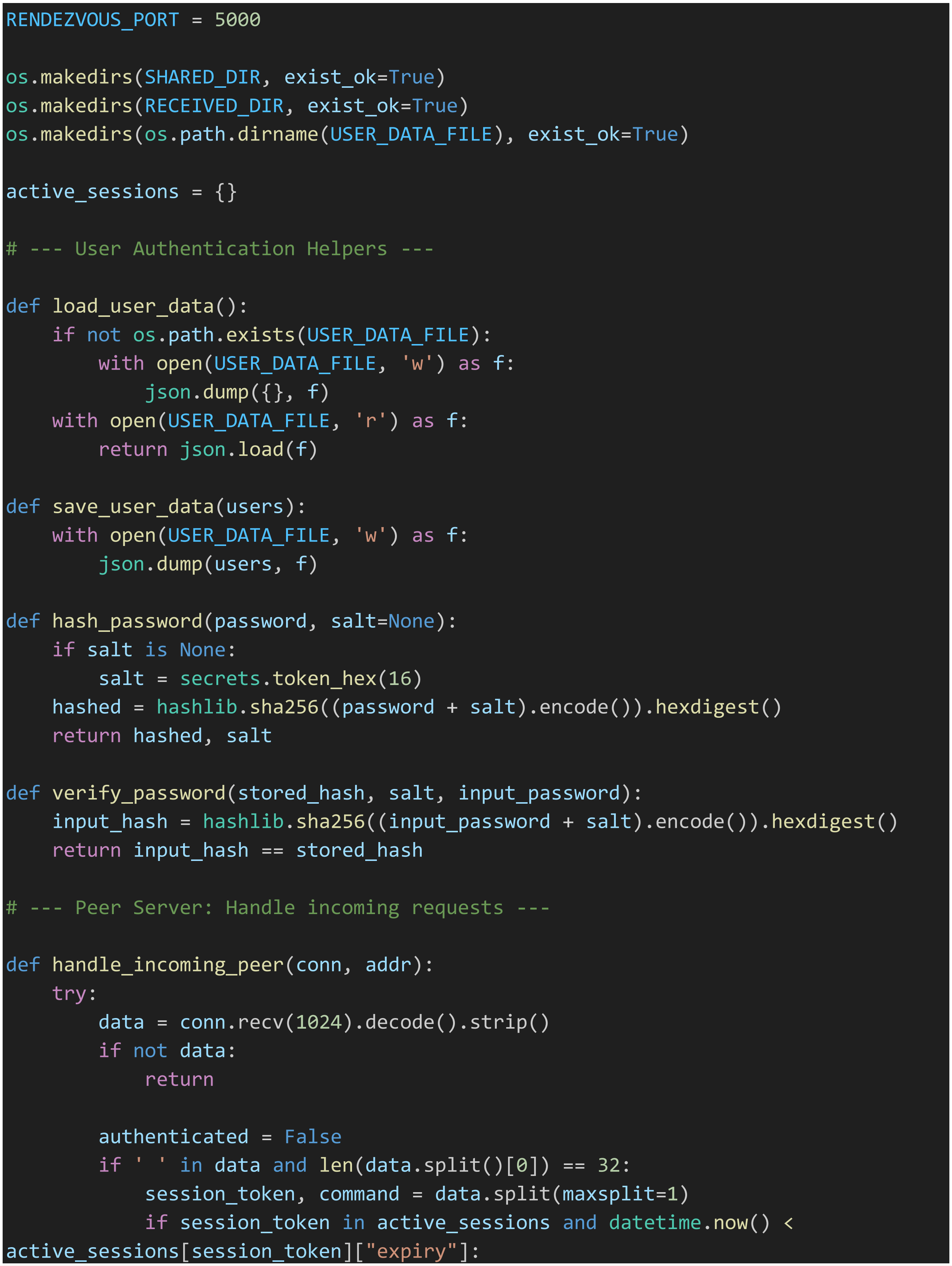


```
                peers.append((peer_ip, peer_port))
                print(f"[+] Peer registered: {peer_ip}:{peer_port}")

            # Send back list of peers (excluding the newly registered peer)
            known_peers = "\n".join(f"{ip} {port}" for ip, port in peers if (ip,
port) != (peer_ip, peer_port))
            conn.sendall(known_peers.encode())
    except Exception as e:
        print(f"[!] Error handling peer: {e}")
    finally:
        conn.close()

def start_rendezvous_server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        print(f"[*] Rendezvous server listening on {HOST}:{PORT}")
        while True:
            conn, addr = s.accept()
            threading.Thread(target=handle_client, args=(conn, addr)).start()

if __name__ == "__main__":
    start_rendezvous_server()
```

File Peer.py:

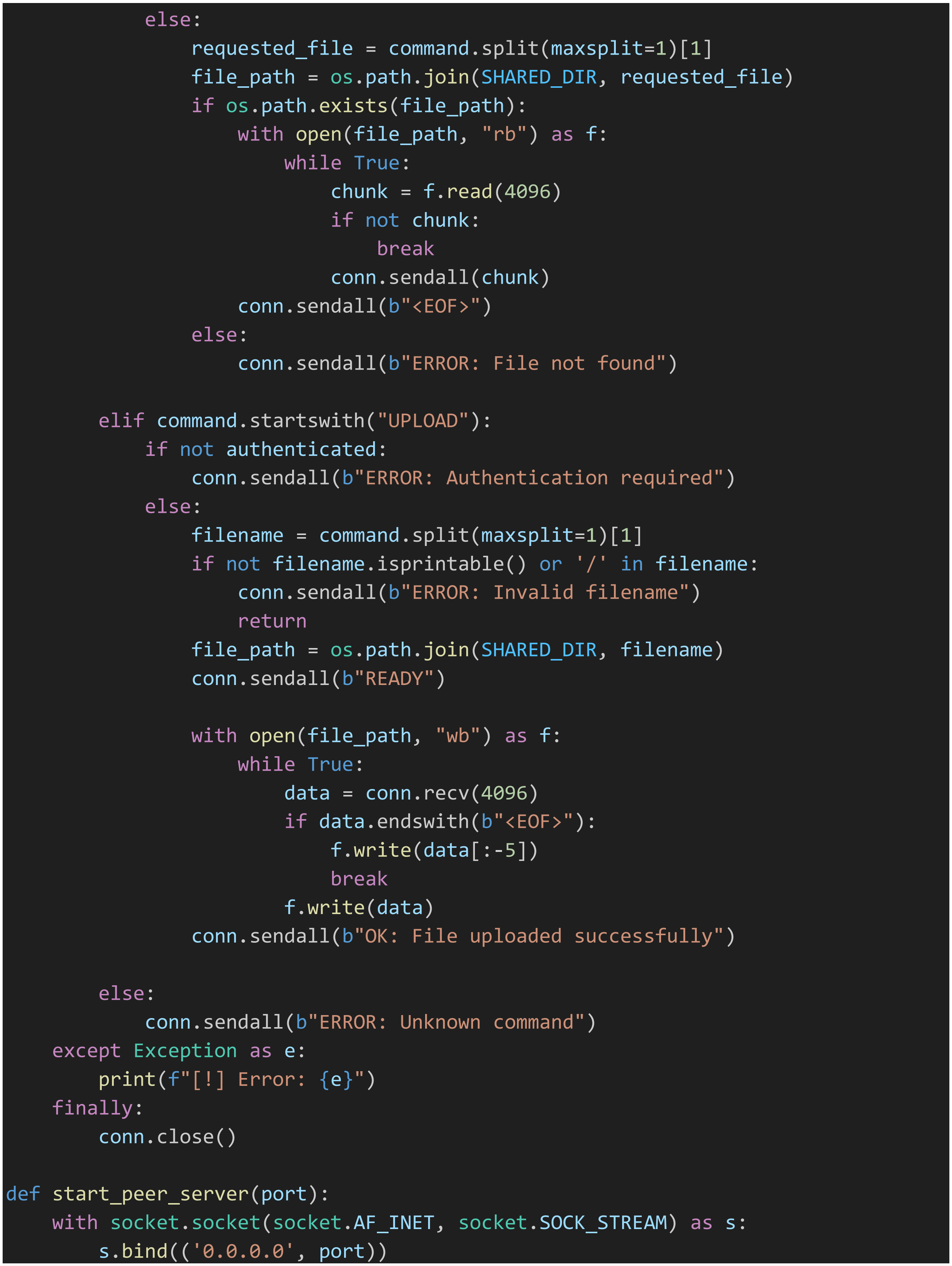


```
# peer.py

import os
import socket
import threading
import hashlib
import secrets
import json
from datetime import datetime, timedelta
import tkinter as tk
from tkinter import filedialog

SHARED_DIR = "shared_files"
RECEIVED_DIR = "received_files"
USER_DATA_FILE = "user_data/users.json"
SESSION_TIMEOUT = 3  # minutes
```

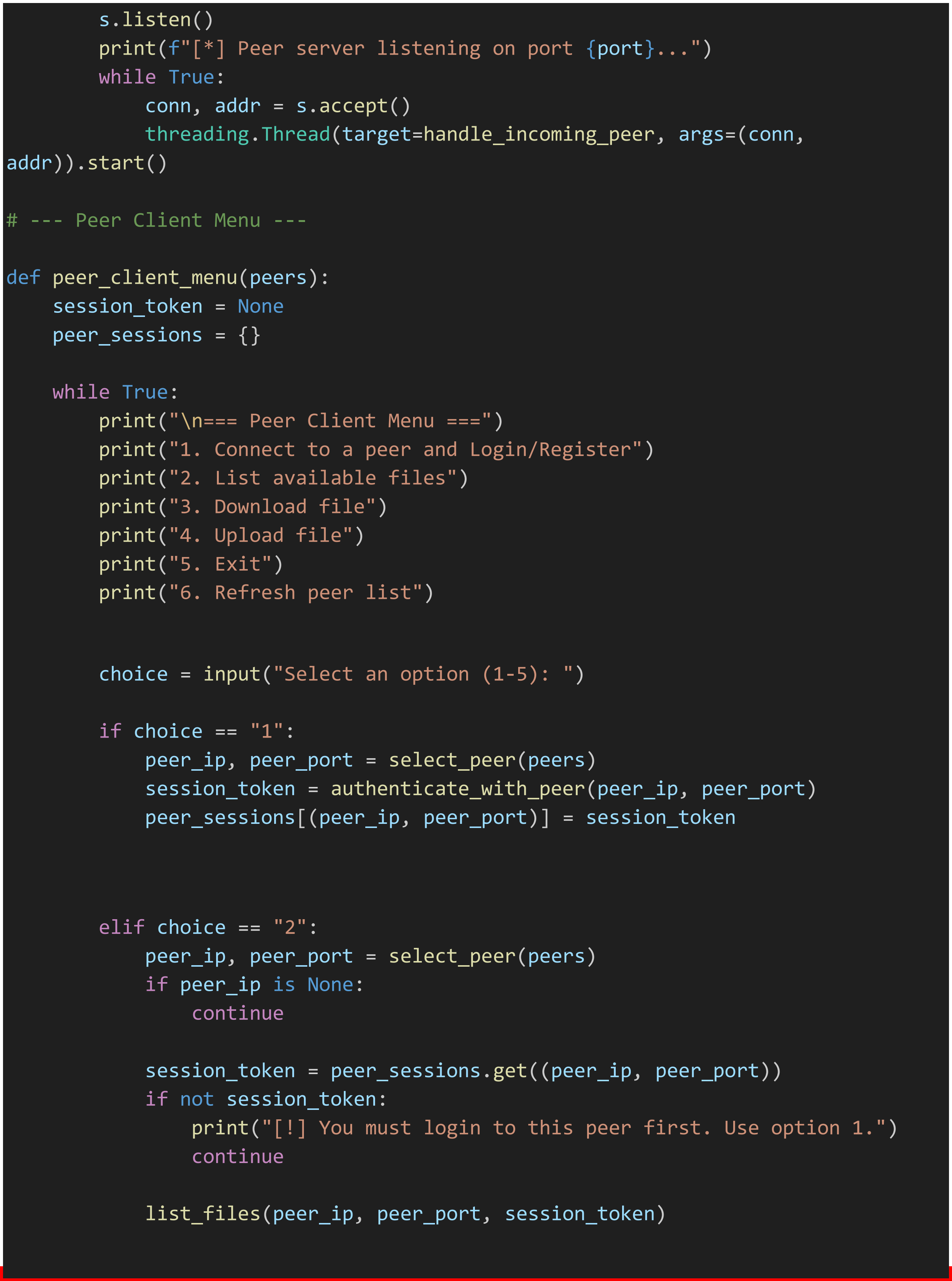


```
RENDEZVOUS_PORT = 5000

os.makedirs(SHARED_DIR, exist_ok=True)
os.makedirs(RECEIVED_DIR, exist_ok=True)
os.makedirs(os.path.dirname(USER_DATA_FILE), exist_ok=True)

active_sessions = {}

# --- User Authentication Helpers ---

def load_user_data():
    if not os.path.exists(USER_DATA_FILE):
        with open(USER_DATA_FILE, 'w') as f:
            json.dump({}, f)
    with open(USER_DATA_FILE, 'r') as f:
        return json.load(f)

def save_user_data(users):
    with open(USER_DATA_FILE, 'w') as f:
        json.dump(users, f)

def hash_password(password, salt=None):
    if salt is None:
        salt = secrets.token_hex(16)
    hashed = hashlib.sha256((password + salt).encode()).hexdigest()
    return hashed, salt

def verify_password(stored_hash, salt, input_password):
    input_hash = hashlib.sha256((input_password + salt).encode()).hexdigest()
    return input_hash == stored_hash

# --- Peer Server: Handle incoming requests ---

def handle_incoming_peer(conn, addr):
    try:
        data = conn.recv(1024).decode().strip()
        if not data:
            return

        authenticated = False
        if ' ' in data and len(data.split()[0]) == 32:
            session_token, command = data.split(maxsplit=1)
            if session_token in active_sessions and datetime.now() <
active_sessions[session_token]["expiry"]:
```

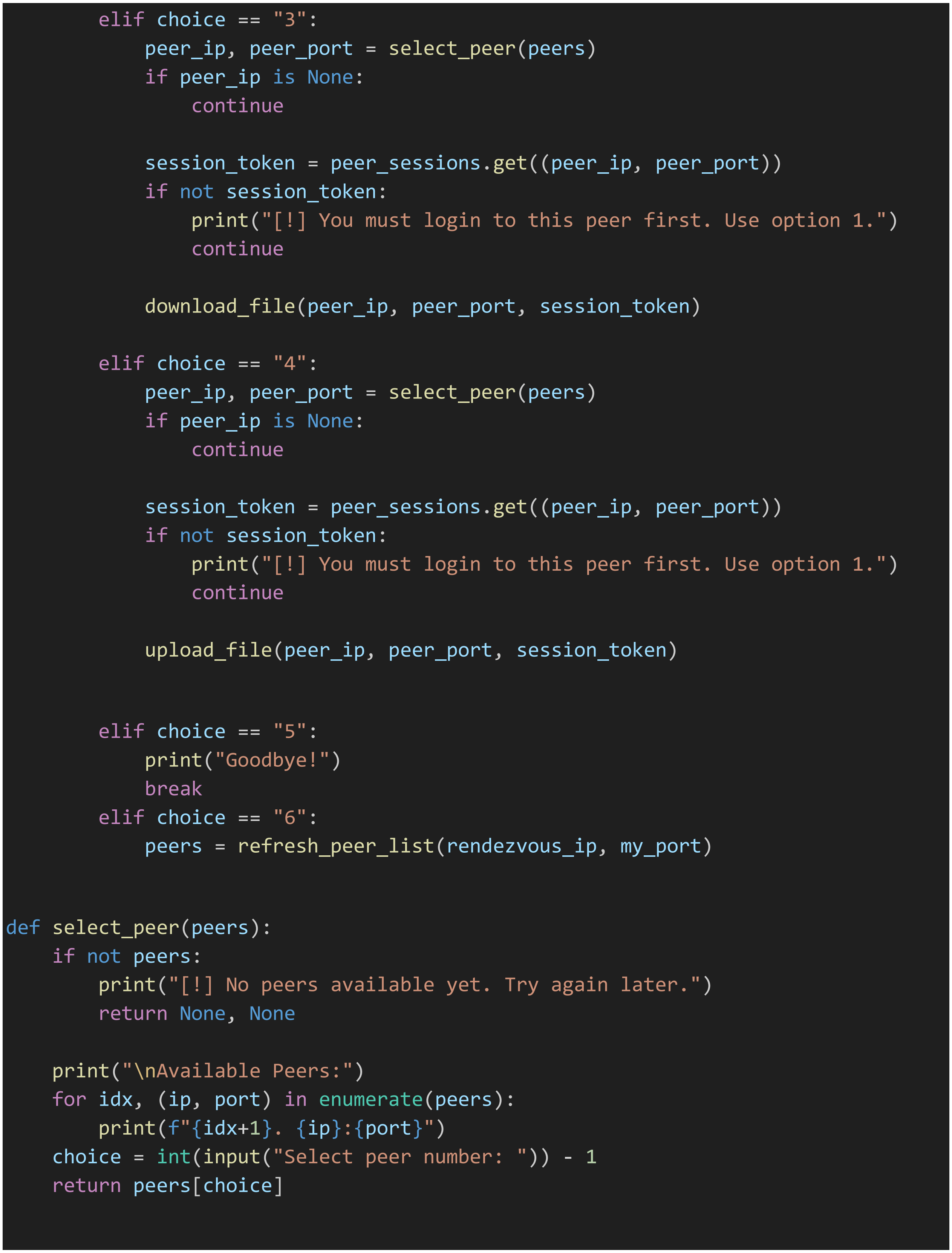
```
                authenticated = True
                # Renew session
                active_sessions[session_token]["expiry"] = datetime.now() + 
timedelta(minutes=SESSION_TIMEOUT)
        else:
            command = data

        users = load_user_data()

        if command.startswith("REGISTER"):
            _, username, password = command.split(maxsplit=2)
            if username in users:
                conn.sendall(b"ERROR: Username already exists")
            else:
                password_hash, salt = hash_password(password)
                users[username] = {"password_hash": password_hash, "salt": salt}
                save_user_data(users)
                conn.sendall(b"OK: Registration successful")

        elif command.startswith("LOGIN"):
            _, username, password = command.split(maxsplit=2)
            if username not in users:
                conn.sendall(b"ERROR: User not found")
            else:
                user_data = users[username]
                if verify_password(user_data["password_hash"], user_data["salt"], 
password):
                    token = secrets.token_hex(16)
                    active_sessions[token] = {"username": username, "expiry": 
datetime.now() + timedelta(minutes=SESSION_TIMEOUT)}
                    conn.sendall(f"OK: {token}".encode())
                else:
                    conn.sendall(b"ERROR: Invalid password")

        elif command == "LIST_FILES":
            if not authenticated:
                conn.sendall(b"ERROR: Authentication required")
            else:
                files = os.listdir(SHARED_DIR)
                conn.sendall("\n".join(files).encode())

        elif command.startswith("DOWNLOAD"):
            if not authenticated:
                conn.sendall(b"ERROR: Authentication required")
```



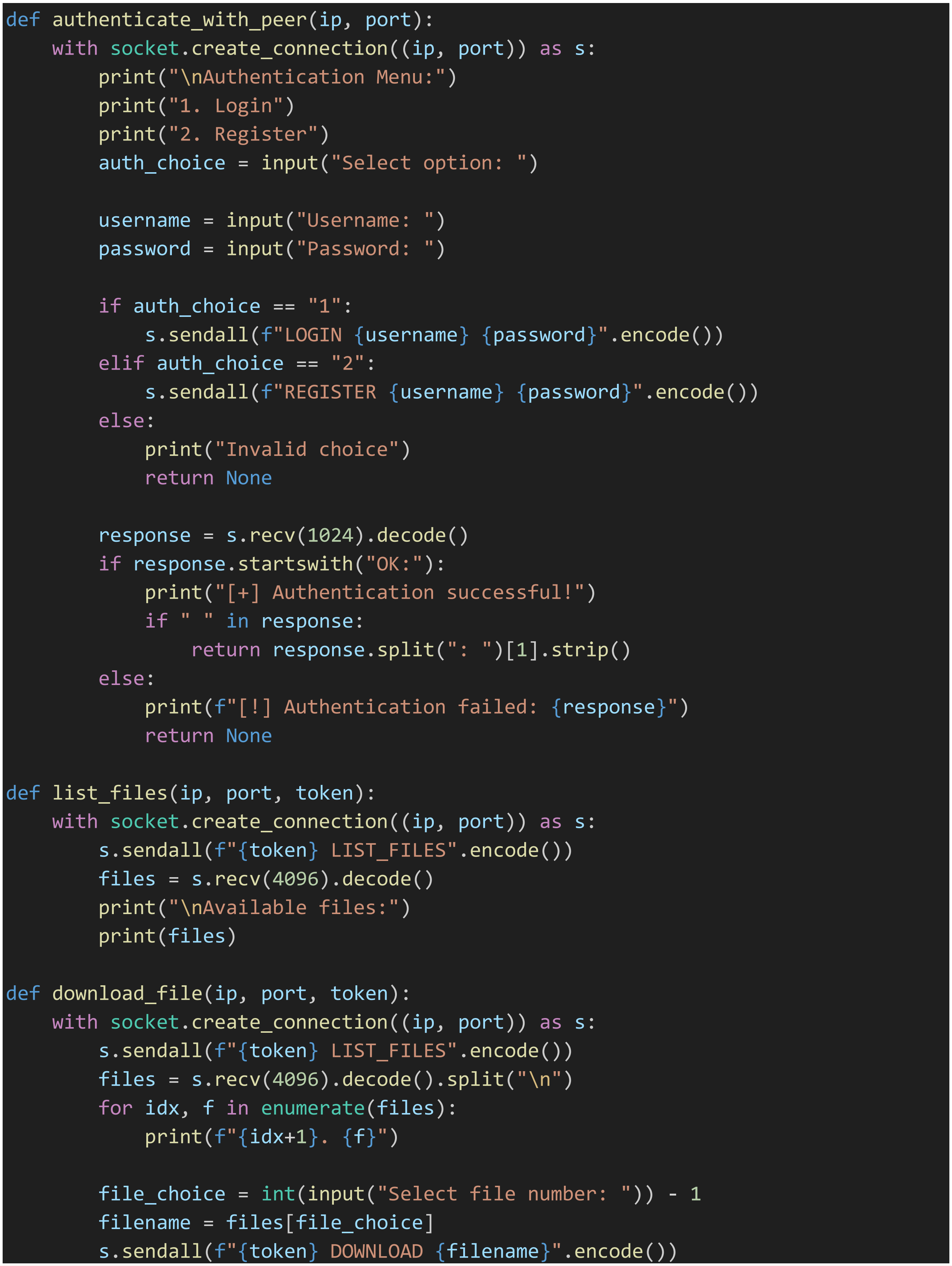
```
            else:
                requested_file = command.split(maxsplit=1)[1]
                file_path = os.path.join(SHARED_DIR, requested_file)
                if os.path.exists(file_path):
                    with open(file_path, "rb") as f:
                        while True:
                            chunk = f.read(4096)
                            if not chunk:
                                break
                            conn.sendall(chunk)
                    conn.sendall(b"<EOF>")
                else:
                    conn.sendall(b"ERROR: File not found")

        elif command.startswith("UPLOAD"):
            if not authenticated:
                conn.sendall(b"ERROR: Authentication required")
            else:
                filename = command.split(maxsplit=1)[1]
                if not filename.isprintable() or '/' in filename:
                    conn.sendall(b"ERROR: Invalid filename")
                    return
                file_path = os.path.join(SHARED_DIR, filename)
                conn.sendall(b"READY")

                with open(file_path, "wb") as f:
                    while True:
                        data = conn.recv(4096)
                        if data.endswith(b"<EOF>"):
                            f.write(data[:-5])
                            break
                        f.write(data)
                conn.sendall(b"OK: File uploaded successfully")

        else:
            conn.sendall(b"ERROR: Unknown command")
    except Exception as e:
        print(f"[!] Error: {e}")
    finally:
        conn.close()

def start_peer_server(port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind(('0.0.0.0', port))
```



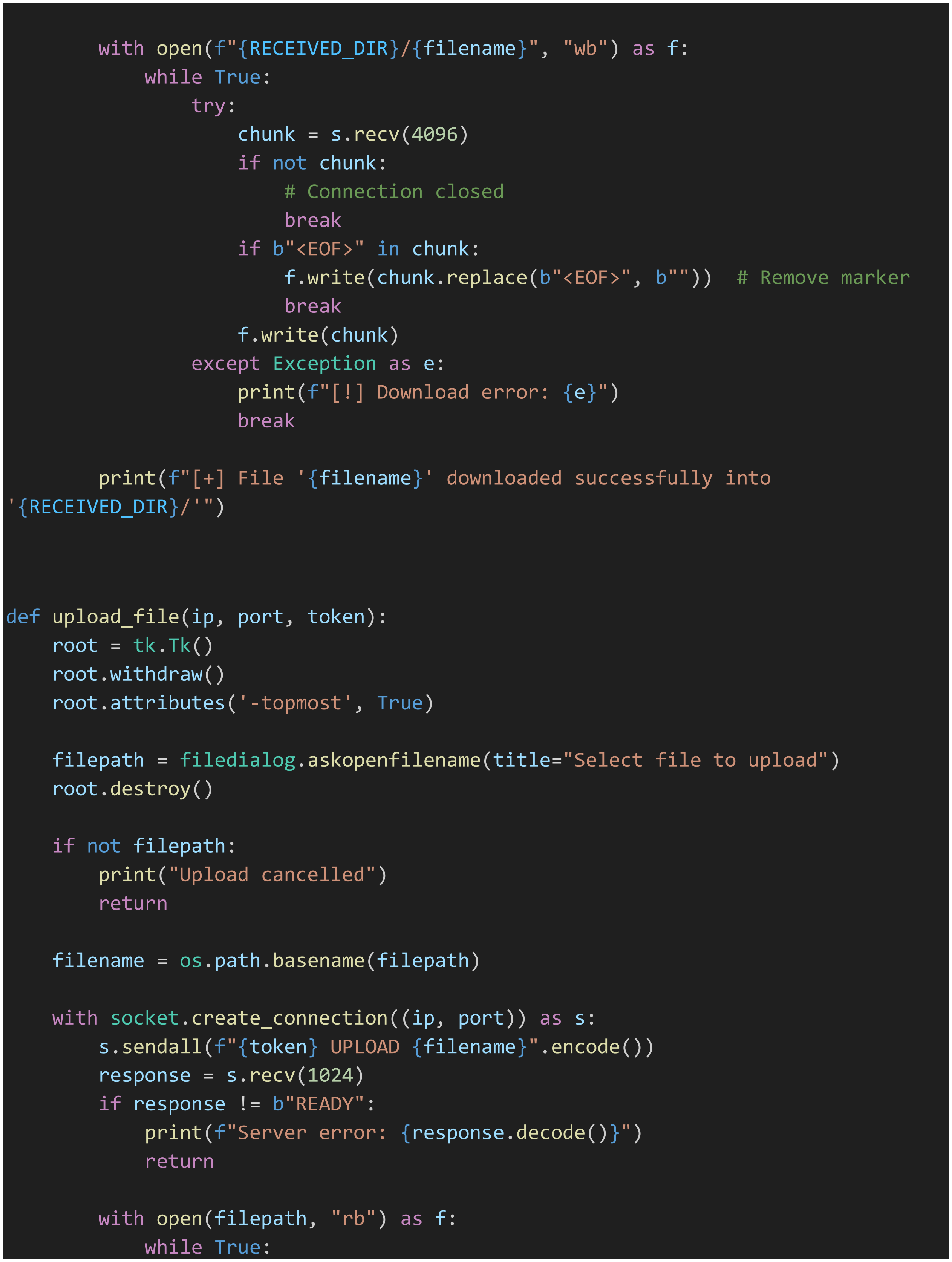
```
        s.listen()
        print(f"[*] Peer server listening on port {port}...")
        while True:
            conn, addr = s.accept()
            threading.Thread(target=handle_incoming_peer, args=(conn,
addr)).start()

# --- Peer Client Menu ---

def peer_client_menu(peers):
    session_token = None
    peer_sessions = {}

    while True:
        print("\n=== Peer Client Menu ===")
        print("1. Connect to a peer and Login/Register")
        print("2. List available files")
        print("3. Download file")
        print("4. Upload file")
        print("5. Exit")
        print("6. Refresh peer list")


        choice = input("Select an option (1-5): ")

        if choice == "1":
            peer_ip, peer_port = select_peer(peers)
            session_token = authenticate_with_peer(peer_ip, peer_port)
            peer_sessions[(peer_ip, peer_port)] = session_token



        elif choice == "2":
            peer_ip, peer_port = select_peer(peers)
            if peer_ip is None:
                continue

            session_token = peer_sessions.get((peer_ip, peer_port))
            if not session_token:
                print("[!] You must login to this peer first. Use option 1.")
                continue

            list_files(peer_ip, peer_port, session_token)
```



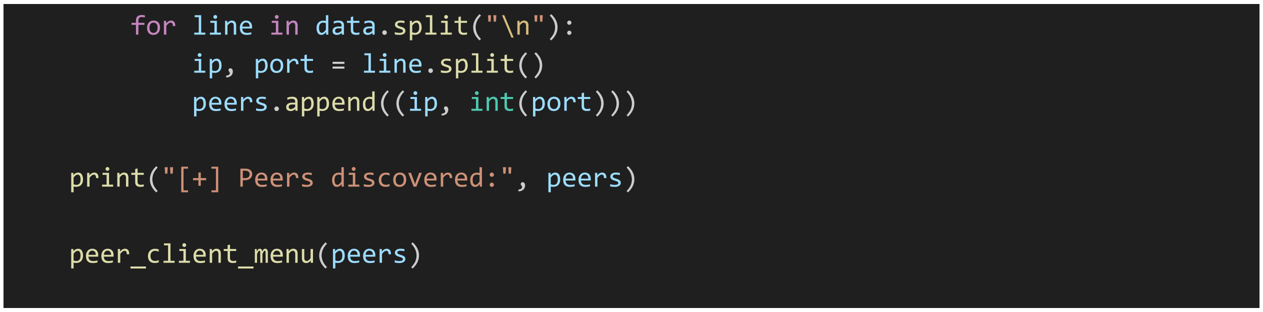
```
        elif choice == "3":
            peer_ip, peer_port = select_peer(peers)
            if peer_ip is None:
                continue

            session_token = peer_sessions.get((peer_ip, peer_port))
            if not session_token:
                print("[!] You must login to this peer first. Use option 1.")
                continue

            download_file(peer_ip, peer_port, session_token)

        elif choice == "4":
            peer_ip, peer_port = select_peer(peers)
            if peer_ip is None:
                continue

            session_token = peer_sessions.get((peer_ip, peer_port))
            if not session_token:
                print("[!] You must login to this peer first. Use option 1.")
                continue

            upload_file(peer_ip, peer_port, session_token)


        elif choice == "5":
            print("Goodbye!")
            break
        elif choice == "6":
            peers = refresh_peer_list(rendezvous_ip, my_port)


def select_peer(peers):
    if not peers:
        print("[!] No peers available yet. Try again later.")
        return None, None

    print("\nAvailable Peers:")
    for idx, (ip, port) in enumerate(peers):
        print(f"{idx+1}. {ip}:{port}")
    choice = int(input("Select peer number: ")) - 1
    return peers[choice]
```



```python
def authenticate_with_peer(ip, port):
    with socket.create_connection((ip, port)) as s:
        print("\nAuthentication Menu:")
        print("1. Login")
        print("2. Register")
        auth_choice = input("Select option: ")

        username = input("Username: ")
        password = input("Password: ")

        if auth_choice == "1":
            s.sendall(f"LOGIN {username} {password}".encode())
        elif auth_choice == "2":
            s.sendall(f"REGISTER {username} {password}".encode())
        else:
            print("Invalid choice")
            return None

        response = s.recv(1024).decode()
        if response.startswith("OK:"):
            print("[+] Authentication successful!")
            if " " in response:
                return response.split(": ")[1].strip()
        else:
            print(f"[!] Authentication failed: {response}")
            return None

def list_files(ip, port, token):
    with socket.create_connection((ip, port)) as s:
        s.sendall(f"{token} LIST_FILES".encode())
        files = s.recv(4096).decode()
        print("\nAvailable files:")
        print(files)

def download_file(ip, port, token):
    with socket.create_connection((ip, port)) as s:
        s.sendall(f"{token} LIST_FILES".encode())
        files = s.recv(4096).decode().split("\n")
        for idx, f in enumerate(files):
            print(f"{idx+1}. {f}")

        file_choice = int(input("Select file number: ")) - 1
        filename = files[file_choice]
        s.sendall(f"{token} DOWNLOAD {filename}".encode())
```



```
        with open(f"{RECEIVED_DIR}/{filename}", "wb") as f:
            while True:
                try:
                    chunk = s.recv(4096)
                    if not chunk:
                        # Connection closed
                        break
                    if b"<EOF>" in chunk:
                        f.write(chunk.replace(b"<EOF>", b""))  # Remove marker
                        break
                    f.write(chunk)
                except Exception as e:
                    print(f"[!] Download error: {e}")
                    break

        print(f"[+] File '{filename}' downloaded successfully into
'{RECEIVED_DIR}/'")



def upload_file(ip, port, token):
    root = tk.Tk()
    root.withdraw()
    root.attributes('-topmost', True)

    filepath = filedialog.askopenfilename(title="Select file to upload")
    root.destroy()

    if not filepath:
        print("Upload cancelled")
        return

    filename = os.path.basename(filepath)

    with socket.create_connection((ip, port)) as s:
        s.sendall(f"{token} UPLOAD {filename}".encode())
        response = s.recv(1024)
        if response != b"READY":
            print(f"Server error: {response.decode()}")
            return

        with open(filepath, "rb") as f:
            while True:
                chunk = f.read(4096)
                if not chunk:
                    break
                s.sendall(chunk)
        s.sendall(b"<EOF>")

        final_response = s.recv(1024).decode()
        print(final_response)

def refresh_peer_list(rendezvous_ip, my_port):
    new_peers = []
    try:
        with socket.create_connection((rendezvous_ip, 5000)) as s:
            my_ip = socket.gethostbyname(socket.gethostname())
            s.sendall(f"REGISTER {my_ip} {my_port}".encode())
            data = s.recv(4096).decode()
            if data.strip():
                for line in data.split("\n"):
                    ip, port = line.split()
                    new_peers.append((ip, int(port)))
    except Exception as e:
        print(f"[!] Could not refresh peers: {e}")

    print("[+] Peers refreshed:", new_peers)
    return new_peers


# --- Main ---

if __name__ == "__main__":
    my_port = int(input("Enter your peer server port: "))
    rendezvous_ip = input("Enter Rendezvous Server IP: ")

    threading.Thread(target=start_peer_server, args=(my_port,),
daemon=True).start()

    my_ip = socket.gethostbyname(socket.gethostname())

    with socket.create_connection((rendezvous_ip, RENDEZVOUS_PORT)) as s:
        s.sendall(f"REGISTER {my_ip} {my_port}".encode())
        data = s.recv(4096).decode()

    peers = []
    if data.strip():
```



```
        for line in data.split("\n"):
            ip, port = line.split()
            peers.append((ip, int(port)))

    print("[+] Peers discovered:", peers)

    peer_client_menu(peers)
```