```python
# 2024 Â© Idan Hazay protocol.py
# Import libraries

from modules.config import *
from modules.file_send import File
from modules import helper, dialogs
from PyQt6.QtGui import QIcon
from PyQt6.QtWidgets import QFileDialog, QApplication, QTableWidgetItem, QHeaderView
import time, socket

class Protocol:
    """Handles client-server communication, including authentication, file management, and user operations."""
    def __init__(self, network, window):
        self.network = network
        self.window = window
        self.ip = SAVED_IP
        self.port = SAVED_PORT

    def change_share(self):
        """Toggles shared files mode and updates directory view."""
        self.window.share = not self.window.share
        self.move_dir("")

    def change_deleted(self):
        """Toggles deleted files mode and updates directory view."""
        self.window.deleted = not self.window.deleted
        self.move_dir("")

    def view_file(self, file_id, file_name, size):
        """Requests a file preview from the server and stores it for viewing."""
        self.send_data(b"VIEW|" + file_id.encode())
        save_path = f"{os.getcwd()}\\temp-{file_name}"
        self.window.files_downloading[file_id] = File(self.window, save_path, file_id, size, True, file_name=file_name)

    def get_file_progress(self):
        """Requests upload progress of all active file uploads."""
        uploading_files = self.window.json.get_files_uploading_data()
        if uploading_files is None:
            return
        for file_id in uploading_files:
            self.send_data(f"RESU|{file_id}".encode())

    def request_resume_download(self):
        """Requests resume points for all interrupted downloads."""
        downloading_files = self.window.json.get_files_downloading_data()
        if downloading_files is None:
            return
        for file_id, details in downloading_files.items():
            file_path = details.get("file_path")
            if not os.path.exists(file_path):
                continue
            progress = details.get("progress")
            self.send_data(f"RESD|{file_id}|{progress}".encode())
            self.window.files_downloading[file_id] = File(self.window, file_path, file_id, details.get("size"),
file_name=details.get("file_name"))

    def send_cookie(self):
        """Sends stored user authentication cookie to the server."""
        try:
            with open(COOKIE_PATH, "r") as f:
                cookie = f.read()
                self.send_data(b"COKE|" + cookie.encode())
        except:
            print("Cookie file not found")

    def get_cwd_files(self, filter=None):
        """Requests the list of files in the current directory."""
        self.get_files(1, filter)

    def get_cwd_shared_files(self, filter=None):
        """Requests the list of shared files."""
        self.get_files(2, filter)

    def get_deleted_files(self, filter=None):
        """Requests the list of deleted files."""
        self.get_files(3, filter)

    def get_cwd_directories(self, filter=None):
        """Requests the list of directories in the current path."""
        self.get_files(4, filter)

    def get_cwd_shared_directories(self, filter=None):
        """Requests the list of shared directories."""
        self.get_files(5, filter)

    def get_deleted_directories(self, filter=None):
        """Requests the list of deleted directories."""
```

```
        self.get_files(6, filter)

    def get_files(self, type, filter):
        """Sends a request to retrieve file and directory information."""
        get_types = {1: ["GETP", "PATH"], 2: ["GESP", "PASH"], 3: ["GEDP", "PADH"],
                     4: ["GETD", "PATD"], 5: ["GESD", "PASD"], 6: ["GEDD", "PADD"]}
        to_send = f"{get_types[type][0]}|{self.window.user['cwd']}|{self.window.current_files_amount}|{self.window.sort}|
{self.window.sort_direction}".encode()
        if filter:
            to_send += b"|" + filter.encode()
        self.send_data(to_send)

    def send_share_premissions(self, dialog, file_id, user_cred, read, write, delete, rename, download, share):
        """Sends updated file-sharing permissions to the server."""
        dialog.accept()
        to_send = f"SHRP|{file_id}|{user_cred}|{read}|{write}|{delete}|{rename}|{download}|{share}"
        self.send_data(to_send.encode())

    def change_username(self):
        """Prompts the user for a new username and sends the request to the server."""
        name = self.window.user["username"]
        new_name = dialogs.new_name_dialog("Change Username", "Enter new username:", name)
        if new_name and new_name != name:
            self.send_data(b"CHUN|" + new_name.encode())

    def subscribe(self, level):
        """Requests a subscription level upgrade."""
        self.send_data(b"SUBL|" + str(level).encode())

    def move_dir(self, new_dir):
        """Requests to navigate to a new directory."""
        self.send_data(f"MOVD|{new_dir}".encode())

    def get_user_icon(self):
        """Requests the user's profile picture from the server."""
        self.send_data(b"GICO")
        self.window.files_downloading["user"] = File(self.window, USER_ICON, "user", 0, file_name="User Icon")

    def get_used_storage(self):
        """Requests the user's current storage usage."""
        self.send_data(b"GEUS")

    def login(self, cred, password, remember_temp):
        """Sends login credentials to the server."""
        self.window.remember = remember_temp
        send_string = helper.build_req_string("LOGN", [cred, password])
        self.send_data(send_string)

    def logout(self):
        """Sends a logout request to the server."""
        send_string = helper.build_req_string("LOGU")
        self.send_data(send_string)

    def signup(self, email, username, password, confirm_password):
        """Sends a new user registration request to the server."""
        send_string = helper.build_req_string("SIGU", [email, username, password, confirm_password])
        self.send_data(send_string)

    def reset_password(self, email):
        """Requests a password reset email from the server."""
        send_string = helper.build_req_string("FOPS", [email])
        self.send_data(send_string)

    def password_recovery(self, email, code, new_password, confirm_new_password):
        """Sends password reset confirmation with new credentials."""
        send_string = helper.build_req_string("PASR", [email, code, new_password, confirm_new_password])
        self.send_data(send_string)


    def send_verification(self, email):
        """Requests a verification email for the user."""
        send_string = helper.build_req_string("SVER", [email])
        self.send_data(send_string)

    def verify(self, email, code):
        """Sends a verification code to the server for confirmation."""
        send_string = helper.build_req_string("VERC", [email, code])
        self.send_data(send_string)

    def delete_user(self, email):
        """Requests account deletion after user confirmation."""
        if self.window.confirm_account_deletion(email):
            send_string = helper.build_req_string("DELU", [email])
            self.send_data(send_string)

    def view_file(self, file_id, file_name, size):
        """Requests a file preview and stores it for viewing."""
```

```
        self.send_data(b"VIEW|" + file_id.encode())
        save_path = f"{os.getcwd()}\\temp-{file_name}"
        self.window.files_downloading[file_id] = File(self.window, save_path, file_id, size, True, file_name=file_name)

    def end_view(self, file_id):
        """Stops viewing a file and releases resources."""
        self.send_data(b"VIEE|" + file_id.encode())

    def update_userpage(self, msg):
        """Updates the user page with a new message."""
        self.send_data(f"UPDT|{msg}".encode())

    def exit_program(self):
        """Sends an exit request to the server."""
        send_string = helper.build_req_string("EXIT")
        self.send_data(send_string)

    def upload_icon(self):
        """Allows the user to select and upload a profile picture."""
        try:
            file_path, _ = QFileDialog.getOpenFileName(self.window, "Open File", "", "Image Files (*.png *.jpg *.jpeg
*.bmp *.gif *.ico);")
            if file_path:
                self.window.file_sending.file_queue.append(file_path)
                self.window.file_sending.send_files("ICOS")
        except:
            print(traceback.format_exc())

    def download(self):
        """Handles file download requests, supporting both single and multiple files."""
        if len(self.window.currently_selected) == 1:
            btn = self.window.currently_selected[0]
            file_name = btn.text().split(" | ")[0][1:]
            file_type = "Zip Files (*.zip);;All Files (*)" if btn.is_folder else "Text Files (*.txt);;All Files (*)"
            file_path, _ = QFileDialog.getSaveFileName(self.window, "Save File", file_name, file_type)
            if file_path:
                self.send_data(b"DOWN|" + btn.id.encode())
                self.window.files_downloading[btn.id] = File(self.window, file_path, btn.id, btn.file_size,
file_name=file_name)
                self.window.json.update_json(False, btn.id, file_path, file=self.window.files_downloading[btn.id],
progress=0)
                try: self.window.file_upload_progress.show()
                except: pass
        else:
            file_path, _ = QFileDialog.getSaveFileName(self.window, "Save File", "", "Zip Files (*.zip);;All Files (*)")
            if file_path:
                name = file_path.split("/")[-1]
                ids = "~".join(btn.id for btn in self.window.currently_selected)
                size = sum(btn.file_size for btn in self.window.currently_selected)
                self.send_data(f"DOWN|{ids}|{name}".encode())
                self.window.files_downloading[ids] = File(self.window, file_path, ids, size, file_name=name)
                self.window.json.update_json(False, ids, file_path, file=self.window.files_downloading[ids], progress=0)
                try: self.window.file_upload_progress.show()
                except: pass

    def delete(self):
        """Deletes selected files after confirmation."""
        if dialogs.show_confirmation_dialog(f"Are you sure you want to delete {len(self.window.currently_selected)}
files?"):
            for btn in self.window.currently_selected:
                self.send_data(b"DELF|" + btn.id.encode())
            self.update_userpage(f"Successfully deleted {len(self.window.currently_selected)} files")
            self.window.currently_selected = []

    def share_action(self):
        """Initiates file-sharing with another user."""
        user_email = dialogs.new_name_dialog("Share", f"Enter email/username of the user you want to share
{len(self.window.currently_selected)} files with:")
        if user_email:
            for btn in self.window.currently_selected:
                self.send_data(b"SHRS|" + btn.id.encode() + b"|" + user_email.encode())
            self.update_userpage(f"Successfully shared {len(self.window.currently_selected)} files")

    def remove(self):
        """Removes shared files from the recipient."""
        for btn in self.window.currently_selected:
            self.send_data(B"SHRE|" + btn.id.encode())
        self.update_userpage(f"Successfully removed {len(self.window.currently_selected)} files from share")
        self.window.currently_selected = []

    def recover(self):
        """Restores deleted files from the trash."""
        for btn in self.window.currently_selected:
            self.send_data(b"RECO|" + btn.id.encode())
        self.update_userpage(f"Successfully recovered {len(self.window.currently_selected)} files")
        self.window.currently_selected = []
```

```
def new_folder(self):
    """Creates a new folder in the current directory."""
    new_folder = dialogs.new_name_dialog("New Folder", "Enter new folder name:")
    if new_folder:
        self.send_data(b"NEWF|" + new_folder.encode())

def search(self):
    """Prompts the user to enter a search filter and updates the file list."""
    self.window.search_filter = dialogs.new_name_dialog("Search", "Enter search filter:", self.window.search_filter)
    self.window.user_page()

def admin_data(self):
    """Request administrator data from server"""
    self.send_data(b"ADMN")

def protocol_parse_reply(self, reply):
    """Parses server responses and executes corresponding actions."""
    try:
        to_show = 'Invalid reply from server'
        if reply is None:
            return None

        # Parse the reply and split it based on the protocol separator
        fields = reply.split(b"|")
        code = fields[0].decode()

        if code != "RILD" and code != "RILE":
            fields = reply.decode().split("|")

        if code == 'ERRR':  # Handle server errors
            err_code = int(fields[1])
            self.window.set_error_message(fields[2])

            if err_code == 9:
                self.window.send_verification_page()
            elif err_code == 14:
                try:
                    file_id = fields[3]
                    self.window.json.update_json(True, file_id, "", remove=True)
                except:
                    print(traceback.format_exc())
            elif err_code == 20:
                if self.window.file_sending.active_threads:
                    self.window.file_sending.active_threads[0].running = False
            elif err_code in [22, 26]:
                try:
                    name = fields[3]
                    file_path = f"{os.getcwd()}\\temp-{name}"
                    if os.path.exists(file_path):
                        os.remove(file_path)
                except:
                    pass

            to_show = f'Server returned an error: {fields[1]} {fields[2]}'

        # Handle each response accordingly
        elif code == 'EXTR':  # Server confirmed exit
            to_show = 'Server acknowledged the exit message'

        elif code == 'LOGS':  # Login successful
            email = fields[1]
            username = fields[2]
            to_show = f'Login was successful for user: {username}'
            self.window.search_filter = None
            self.window.user["email"] = email
            self.window.user["username"] = username
            self.window.user["subscription_level"] = fields[3]
            self.window.user["admin_level"] = int(fields[4])
            self.get_user_icon()

            if self.window.user["username"].lower() == "emily":
                with open(f"{os.getcwd()}/gui/css/emily.css", 'r') as f:
                    self.app.setStyleSheet(f.read())

            self.window.user_page()
            self.window.set_message("Login was successful!")
            if self.window.remember:
                self.send_data(b"GENC")

        elif code == 'SIGS':  # Signup completed
            email = fields[1]
            username = fields[2]
            password = fields[3]
            to_show = f'Signup was successful for user: {username}, password: {password}'

            self.window.verification_page(email)
            self.window.set_message(f"Signup for user {username} completed. Verification code sent to your email.")
```

```
            elif code == 'FOPR':  # Password reset email sent
                to_show = f'Password reset code was sent to {fields[1]}'
                self.window.recovery(fields[1])
                self.window.set_message(to_show)

            elif code == 'PASS':  # Password reset successful
                new_pwd = fields[2]
                to_show = f'Password was reset for user: {fields[1]}, new password: {new_pwd}'
                self.logout()
                self.window.main_page()
                self.window.set_message("Password reset successful, please log in again.")

            elif code == 'LUGR':  # Logout confirmed
                if self.window.user["username"].lower() == "emily":
                    with open(f"{os.getcwd()}/gui/css/style.css", 'r') as f:
                        self.app.setStyleSheet(f.read())

                self.window.user.update({"email": "guest", "username": "guest", "subscription_level": 0, "cwd": "",
"parent_cwd": "", "cwd_name": ""})
                self.window.share = False
                self.window.deleted = False
                to_show = 'Logout successful'
                self.window.main_page()
                self.window.set_message(to_show)

            elif code == 'VERS':  # Verification email sent
                email = fields[1]
                to_show = f'Verification sent to email {email}'
                self.window.verification_page(email)
                self.window.set_message(f'Verification email sent to {email}')

            elif code == 'VERR':  # Verification successful
                username = fields[1]
                to_show = f'Verification for user {username} was successful'

                self.window.main_page()
                self.window.set_message(f"Verification for user {username} completed. You may now log in.")

            elif code == 'DELR':  # User deleted successfully
                username = fields[1]
                to_show = f'User {username} was deleted'
                self.window.main_page()
                self.window.set_message(to_show)

            elif code == 'FILR':  # File upload complete
                to_show = f'File {fields[1]} was uploaded'
                if time.time() - self.window.last_load > 0.5:
                    self.window.user_page()
                    self.window.last_load = time.time()
                self.window.set_message(to_show)

            elif code == 'FISS': # File upload started
                to_show = f'File {fields[1]} started uploading'
                self.window.set_message(to_show)

            elif code == 'MOVR':  # Directory changed successfully
                self.window.user["cwd"], self.window.user["parent_cwd"], self.window.user["cwd_name"] = fields[1],
fields[2], fields[3]
                to_show = f'Successfully moved to {fields[3]}'
                self.window.scroll_progress = 0
                self.window.current_files_amount = ITEMS_TO_LOAD
                self.window.user_page()

            elif code in ["RILD", "RILE"]:  # File chunk received
                file_id = fields[1].decode()
                location_infile = int(fields[2].decode())
                data = reply[4 + len(file_id) + len(str(location_infile)) + 3:]

                if file_id in self.window.files_downloading:
                    self.window.files_downloading[file_id].add_data(data, location_infile)

                if code == "RILE":  # Final file chunk received
                    if file_id in self.window.files_downloading:
                        if self.window.files_downloading[file_id].is_view:
                            self.end_view(file_id)
                            self.window.activate_file_view(file_id)
                        self.window.json.update_json(False, file_id, "", remove=True)
                        self.window.set_message(f"File {self.window.files_downloading[file_id].file_name} finished
downloading")
                        del self.window.files_downloading[file_id]

                    try:
                        self.window.stop_button.setEnabled(False)
                        self.window.stop_button.hide()
                    except:
                        pass
```

```
        try:
            self.window.file_upload_progress.hide()
        except:
            pass

    to_show = "File data received " + str(location_infile + len(data))

elif code == 'DOWR':  # File downloaded
    to_show = f'File {fields[1]} was downloaded'
    self.window.set_message(to_show)

elif code == 'NEFR':  # New folder created
    to_show = f'Folder {fields[1]} was created'
    self.window.user_page()
    self.window.set_message(to_show)

elif code == 'RENR':  # File/folder renamed
    to_show = f'File/Folder {fields[1]} was renamed to {fields[2]}'
    self.window.user_page()
    self.window.set_message(to_show)

elif code == 'GICR':  # Profile picture received
    to_show = "Profile picture was received"
    try:
        if self.window.share or self.window.deleted:
            self.window.upload_button.setIcon((QIcon(USER_ICON)))
        self.window.user_button.setIcon((QIcon(USER_ICON)))
    except:
        pass

elif code == 'ICOR':  # Profile icon upload started
    to_show = "Profile icon upload started successfully!"
    self.window.set_message(to_show)

elif code == 'ICUP':  # Profile icon upload complete
    to_show = "Profile icon uploaded successfully!"
    self.get_user_icon()

elif code == 'DLFR':  # File deleted
    file_name = fields[1]
    to_show = f"File {file_name} was deleted!"
    self.window.set_message(to_show)

elif code == 'DFFR':  # Folder deletion confirmed
    folder_name = fields[1]
    to_show = f"Folder {folder_name} was deleted!"
    self.window.set_message(to_show)

elif code == 'SUBR':  # Subscription level updated
    level = fields[1]
    self.window.user["subscription_level"] = level
    sub = ["free", "basic", "premium", "professional"][int(level)]
    to_show = f"Subscription level updated to {sub}"
    self.window.subscriptions_page()
    self.window.set_message(to_show)

elif code == 'GEUR':  # Storage usage retrieved
    self.window.used_storage = round(int(fields[1]) / 1_000_000, 3)
    self.window.set_used_storage()
    to_show = f"Current used storage is {self.window.used_storage} MB"

elif code == 'CHUR':  # Username changed successfully
    new_username = fields[1]
    self.window.user["username"] = new_username
    to_show = f"Username changed to {new_username}"
    self.window.manage_account()
    self.window.set_message(to_show)

elif code == 'VIER':  # File viewed
    file_name = fields[1]
    to_show = f"File {file_name} was viewed"
    self.window.set_message(to_show)

elif code == 'COOK':  # Cookie received
    cookie = fields[1]
    self.save_cookie(cookie)
    to_show = "Cookie received"

elif code == "SHRR":  # Sharing options retrieved
    file_id, user_cred, file_name = fields[1], fields[2], fields[3]
    if len(fields) == 3:
        self.window.share_file(file_id, user_cred, file_name)
    else:
        self.window.share_file(file_id, user_cred, file_name, *fields[4:])
    to_show = "Sharing options received"

elif code == "SHPR":  # Sharing permissions updated
```

```
                to_show = fields[1]
                self.window.set_message(to_show)

            elif code == "SHRM":  # File removed from shared list
                name = fields[1]
                to_show = f"Successfully removed {name} from share"
                self.window.set_message(to_show)

            elif code == "RECR":  # File recovered from deleted state
                name = fields[1]
                to_show = f"Successfully recovered {name}"
                self.window.set_message(to_show)

            elif code == "UPFR":  # File update completed
                name = fields[1]
                to_show = f"Successfully saved changes to file {name}"
                self.window.user_page()
                self.window.set_message(to_show)

            elif code == "VIRR":  # File viewing session released
                to_show = "File viewing released"

            elif code == "STOR":  # File upload stopped
                name, file_id = fields[1], fields[2]
                to_show = f"Upload of {name} stopped"
                if self.window.file_sending.active_threads:
                    self.window.file_sending.active_threads[0].running = False
                self.window.json.update_json(True, file_id, "", remove=True)
                self.window.set_message(to_show)

            elif code in ["PATH", "PASH", "PADH"]:  # Files list retrieved
                self.window.files = fields[2:]
                if self.window.files is not None and self.window.directories is not None:
                    self.window.update_current_files()
                to_show = "Got files"

            elif code in ["PATD", "PASD", "PADD"]:  # Directories list retrieved
                self.window.items_amount = fields[1]
                self.window.total_files.setText(f"{self.window.items_amount} items")
                self.window.directories = fields[2:]
                if self.window.files is not None and self.window.directories is not None:
                    self.window.update_current_files()
                to_show = "Got directories"

            elif code == "RESR":  # File upload resumed
                file_id, progress = fields[1], fields[2]
                self.window.file_sending.resume_files_upload(file_id, progress)
                to_show = f"File upload of file {file_id} continued at {progress}"

            elif code == "RUSR":  # File download resumed
                file_id, progress = fields[1], fields[2]
                to_show = f"Resumed download of file {file_id} from byte {progress}"

            elif code == "UPDR":  # User page updated
                msg = fields[1]
                self.window.user_page()
                self.window.set_message(msg)
                to_show = msg

            elif code == "ADMR":  # Got admin data
                users_info = fields[1:]
                try:
                    table = self.window.users_table
                    table.setRowCount(len(users_info))  # Set number of rows
                    table.setColumnCount(8)
                    table.setHorizontalHeaderLabels(["Id", "Email", "UserName", "Is Verified", "Sub Lvl", "Admin Lvl",
"Files Amount", "Used Storage"])
                    table.horizontalHeader().setStretchLastSection(True)
                    table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)
                    for i, user in enumerate(users_info):
                        user = user.split("~")
                        if user[3] == "1": user[3] = "True"
                        else: user[3] = "False"
                        user[7] = helper.format_file_size(int(user[7]))
                        for j, item in enumerate(user):
                            table.setItem(i, j, QTableWidgetItem(item))

                except:
                    print(traceback.format_exc())
                to_show = "Got admin data"

            else:  # Unknown server response
                self.window.set_message(f"Unknown command {code}")

            if code not in ["RILD", "RILE"]:
                self.window.force_update_window()
```

```
    except Exception:
        print(traceback.format_exc())
    return to_show

def connect_server(self, new_ip=SAVED_IP, new_port=SAVED_PORT, loop=False):
    """Attempts to connect to the server and establish a secure connection."""
    self.window.set_message(f"Trying to connect to {new_ip} {new_port}...")
    QApplication.processEvents()
    self.network.reset_network()
    self.window.receive_thread.pause()

    try:
        self.ip = new_ip
        try:
            self.port = int(new_port)  # Ensure port is a valid integer
        except:
            return

        sock = socket.socket()
        sock.settimeout(SOCK_TIMEOUT)

        try:
            sock.connect((self.ip, self.port))  # Attempt direct connection
        except TimeoutError:  # If timeout occurs, search for a server
            self.ip, self.port = self.network.search_server()
            sock = socket.socket()
            sock.settimeout(SOCK_TIMEOUT)
            sock.connect((self.ip, self.port))

        try:
            helper.update_saved_ip_port(self.ip, self.port)
        except:
            pass
        self.network.set_sock(sock)

        shared_secret = self.network.encryption.rsa_exchange()  # Perform secure key exchange
        if not shared_secret:
            sock.close()
            return

        self.network.set_secret(shared_secret)
        self.window.main_page()
        self.window.receive_thread.start()
        self.window.receive_thread.resume()
        self.send_cookie()
        self.get_file_progress()
        self.request_resume_download()

        self.window.set_message(f'Connect succeeded {self.ip} {self.port}')
        if self.window.user["username"] != "guest":
            self.window.set_message(f'Auto login with account {self.window.user["username"]}')

        return sock

    except TimeoutError:
        if not loop:
            self.window.receive_thread.pause()
            self.window.not_connected_page()
        self.window.set_error_message(f'Server was not found {self.ip} {self.port}')
        return None

    except:
        print(traceback.format_exc())

def send_data(self, bdata, encryption=True):
    """Sends data to the server with optional encryption."""
    try:
        self.network.send_data_wrap(bdata, encryption)
    except ConnectionResetError:  # Handle sudden disconnection
        self.network.sock.close()
        self.window.not_connected_page()
        self.window.set_error_message("Lost connection to server")
    except:
        print(traceback.format_exc())

@staticmethod
def save_cookie(cookie):
    """Saves the authentication cookie for persistent login."""
    if not os.path.exists(os.getcwd() + "\\cookies"):
        os.makedirs(os.getcwd() + "\\cookies")
    with open(COOKIE_PATH, "w") as f:
        f.write(cookie)
```