Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

# Лабораторна робота №9

## **Технології розроблення програмного забезпечення**

## «*РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE*»

Варіант 29

Виконав:
студент групи ІА-13
Хілько І.А.

Перевірив:
Мягкий М. Ю.

Київ 2023

**Тема:** Різні види взаємодії додатків: CLIENT-SERVER, PEER TO-PEER, SERVICE-ORIENTED ARCHITECTURE.

**Варіант:**
Система для колективних покупок. (State, Chain of responsibility, Abstract factory, Mediator, Composite, **Client-server**).

Система дозволяє створити список групи для колективної покупки, список що потрібно купити з орієнтовною вартістю кожної позиції та орієнтовною загальною вартістю, запланувати хто що буде купляти. Щоб користувач міг відмітити що він купив, за яку суму, з можливістю прикріпити чек. Система дозволяє користувачу вести списки бажаних для нього покупок, з можливістю позначати списки, які будуть доступні для друзів (як списки, що можна подарувати користувачеві). Система дозволяє добавляти інших користувачів в друзі.

**Хід роботи:**
1. Реалізувати взаємодію програми в одній з архітектур.

Кліент-серверна архітектура використовується для забезпечення взаємодії між клієнтом та сервером за допомогою сокетів. Основна ідея полягає в тому, щоб клієнт і сервер могли обмінюватися повідомленнями через мережу.

Сервер обробляє реєстрацію користувача, вхід та різноманітні команди, пов'язані з управлінням групами, елементами, списками бажань та інформацією користувача. Клієнт підключається до сервера та надсилає команди для взаємодії з сервером.

## Серверна частина:

Клас Server:

Керує сервером.

З'єднується з конкретним хостом і портом.

Очікує вхідні з'єднання та створює новий потік для кожного клієнта.

Обробляє команди клієнта на основі стану клієнта.

```
class Server:
    def __init__(self, host='127.0.0.1', port=5559):
        self.host = host
        self.port = port
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.bind((self.host, self.port))
        self.server.listen()
        self.current_state = ServerState.NOT_LOGGED_IN
        self.register_handler = RegisterHandler()
        self.login_handler = LoginHandler()
        self.login_register_handler = LoginRegisterCommandHandler()
        self.client_states = {}
        print(f'Server is listening on {self.host}:{self.port}')
    def start_server(self):
        while True:
            client, address = self.server.accept()
            print(f'Accepted connection from {address}')
            client_handler = threading.Thread(target=self.handle_client,
args=(client,))
            client_handler.start()
```

```
def handle_client(self, client_socket):
        self.client_states[client_socket] = ServerState.NOT_LOGGED_IN
        current_user = None
        while True:
            data = client_socket.recv(1024)
            if not data:
                break
            decoded_data = data.decode('utf-8')
            print(f'Received from client: {decoded_data}')

            if self.client_states[client_socket] == ServerState.NOT_LOGGED_IN:
                if decoded_data.startswith(('REGISTER', 'LOGIN')):
                    command, *command_args = decoded_data.split()
                    current_user = self.login_register_handler.handle(
                        self, client_socket, current_user, command, command_args
                    )
                    if current_user and command == 'LOGIN':
                        self.client_states[client_socket] = ServerState.LOGGED_IN
                else:
                    client_socket.send('Please log in first.'.encode('utf-8'))
            elif self.client_states[client_socket] == ServerState.LOGGED_IN:
                self.process_user_command(client_socket, current_user,
decoded_data.split())

        del self.client_states[client_socket]
        client_socket.close()
```

```
    def process_user_command(self, client_socket, current_user, command_args):
        if command_args[0] == 'CREATE_GROUP':
            self.create_group(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'JOIN_GROUP':
            self.join_group(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'LIST_GROUPS':
            self.list_groups(client_socket)
        elif command_args[0] == 'ADD_ITEM':
            self.add_item(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'LIST_ITEMS_IN_GROUP':
            self.list_items_in_group(client_socket, command_args[1:])
        elif command_args[0] == 'MARK_PURCHASE':
            self.mark_purchase(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'ASSIGN_ITEM':
            self.assign_item(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'ADD_FRIEND':
            self.add_friend(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'USER_INFO':
            self.user_info(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'CREATE_WISHLIST':
            self.create_wishlist(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'MANAGE_WISHLIST_VISIBILITY':
            self.manage_wishlist_visibility(client_socket, current_user,
command_args[1:])
        elif command_args[0] == 'VIEW_WISHLIST':
            self.view_wishlist(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'ADD_TO_WISHLIST':
            self.add_to_wishlist(client_socket, current_user, command_args[1:])
        elif command_args[0] == 'LIST_WISHLISTS':
            self.list_wishlists(client_socket, current_user, command_args[1:])
        else:
            client_socket.send('Invalid command. Please enter a valid
command.'.encode('utf-8'))
```

```
if __name__ == "__main__":
    from database import main, DatabaseManager
    main()
    server_instance = Server()
    server_instance.start_server()
```

## Клієнтська частина:

Створення сокету.

Підключення до сервера.

Приймання введення користувача для команд і надсилання їх на сервер.

Друкування відповіді сервера.

```python
import socket

def create_client_socket():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    return client_socket

def connect_to_server(client_socket, host, port):
    client_socket.connect((host, port))

def send_command(client_socket, command):
    client_socket.sendall(command.encode())

def receive_response(client_socket):
    response = client_socket.recv(1024).decode()
    return response

def main():
    host = "localhost"
    port = 5559

    client_socket = create_client_socket()

    try:
        connect_to_server(client_socket, host, port)

        while True:
            command = input("Enter a command (or 'exit' to quit): ")

            if command.lower() == 'exit':
                break

            send_command(client_socket, command)
            response = receive_response(client_socket)
            print(response)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client_socket.close()

if __name__ == '__main__':
    main()
```

**Висновок:** Отже, під час виконання лабораторної роботи, я реалізував взаємодію програми в одній з архітектур, а саме – Client-server.