

# PROJEKT PSI

## RAPORT KOŃCOWY

Temat projektu: chat tekstowy w architekturze p2p

Mateusz Krakowski

Marcin Łojek

Piotr Zając

Michał Kindeusz

19 stycznia 2023

## 1 Treść zadania

Chat tekstowy używający architekturę peer to peer i wykorzystujący rozgłaszanie w sieci lokalnej. Czyli każdy peer rozgłasza cyklicznie, że istnieje i żyje, a pozostali budują sobie listę aktywnych klientów, którą cyklicznie przeglądają i usuwają nieaktywnych klientów. Sama komunikacja odbywa się bezpośrednio pomiędzy rozmówcami. Można wprowadzić ograniczenie liczby klientów np. do 5, aby nie komplikować zbytnio kodu dynamicznymi strukturami.

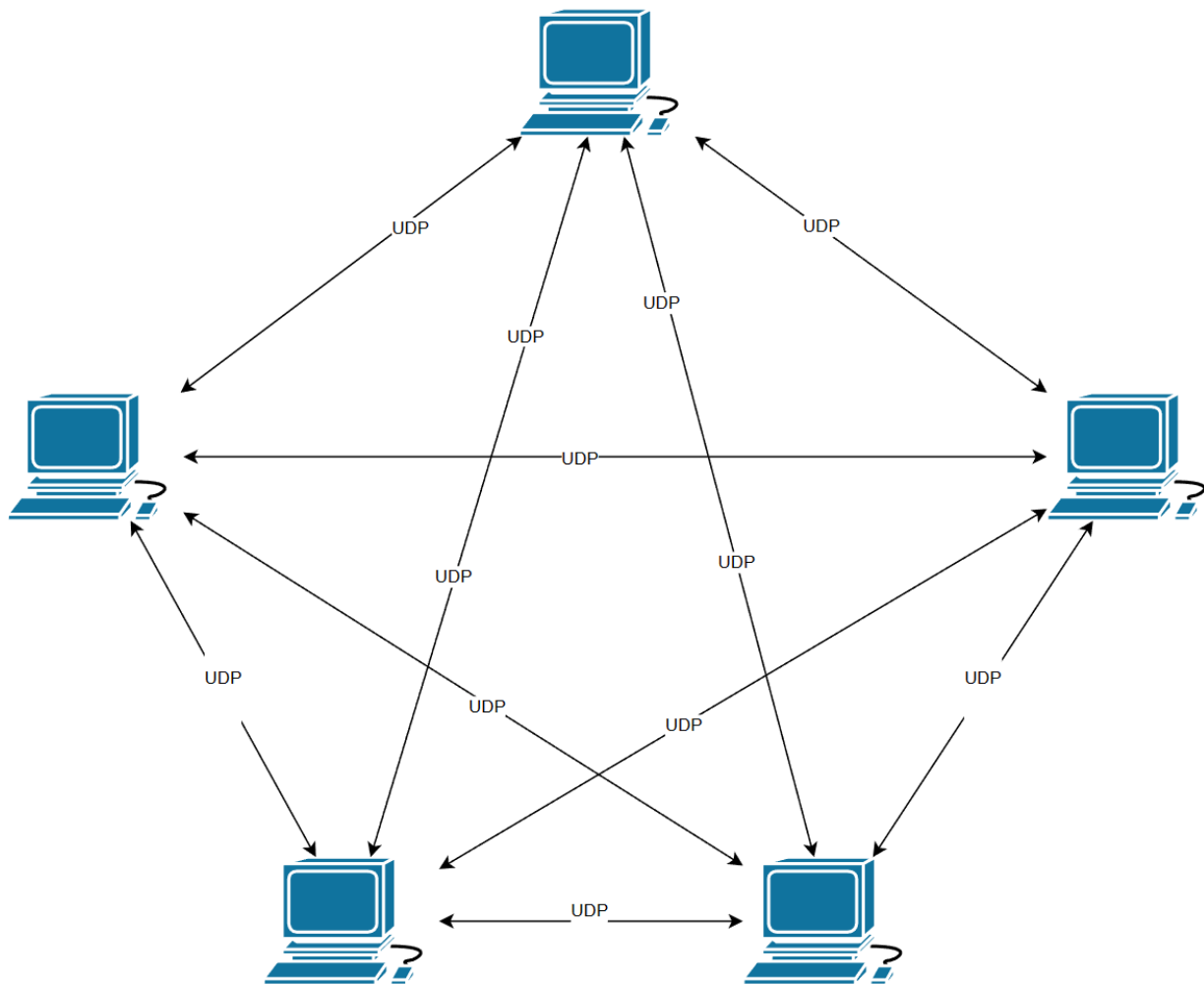
## 2 Opis funkcjonalny black-box

Aplikacja terminalowa realizująca chat grupowy w architekturze peer to peer. Wymagania funkcjonalne:

1. Logowanie się użytkownika za pomocą nicku (domyślnie użytkownicy rozróżniani są przez adres IP)
2. Lista aktualnie dostępnych użytkowników
3. Wysyłanie wiadomości do wszystkich peerów
4. Historia wiadomości tylko dla aktualnej sesji aplikacji, bez historii
5. Brak sztywno narzuconej maksymalnej liczby peerów (Python domyślnie korzysta z dynamicznych struktur, więc nie będzie z tym problemu)

### 3 Opis i analiza poprawności stosowanych protokołów komunikacyjnych

Rozgłaszanie w sieci lokalnej za pomocą protokołu UDP w celu ustalenia listy dostępnych użytkowników



W projekcie postanowiliśmy wykorzystać protokół UDP zarówno do rozgłaszania w sieci lokalnej w celu ustalenia listy dostępnych peerów, jak i samego rozsyłania wiadomości. Dzieje się to z tą różnicą, że PINGI wysyłane są na adres rozgłoszeniowy (czyli do wszystkich peerów w sieci lokalnej), natomiast samo wysyłanie wiadomości jest wykonywane do wszystkich peerów z osobna (znamy ich adresy dzięki wcześniejszemu ustaleniu listy dostępnych peerów).

Flowchart działania peera znajduje się w oddzielnym pliku `flowchartaplikacji.svg`, opis w sekcji "zarys koncepcji implementacji".

## 4 Planowany podział na moduły i strukturę komunikacji między nimi

Podział na moduły:

1. peer.py

- funkcja informująca innych podłączonych klientów o aktywności użytkownika
- funkcja do odbierania wiadomości od innych użytkowników
- funkcja do rozsyłania wiadomości do reszty użytkowników

2. ui.py

- funkcja do pobierania danych wejściowych od użytkownika
- funkcja do wyświetlania wiadomości wymienionych z danym użytkownikiem

3. client.py

- definicja klasy Client

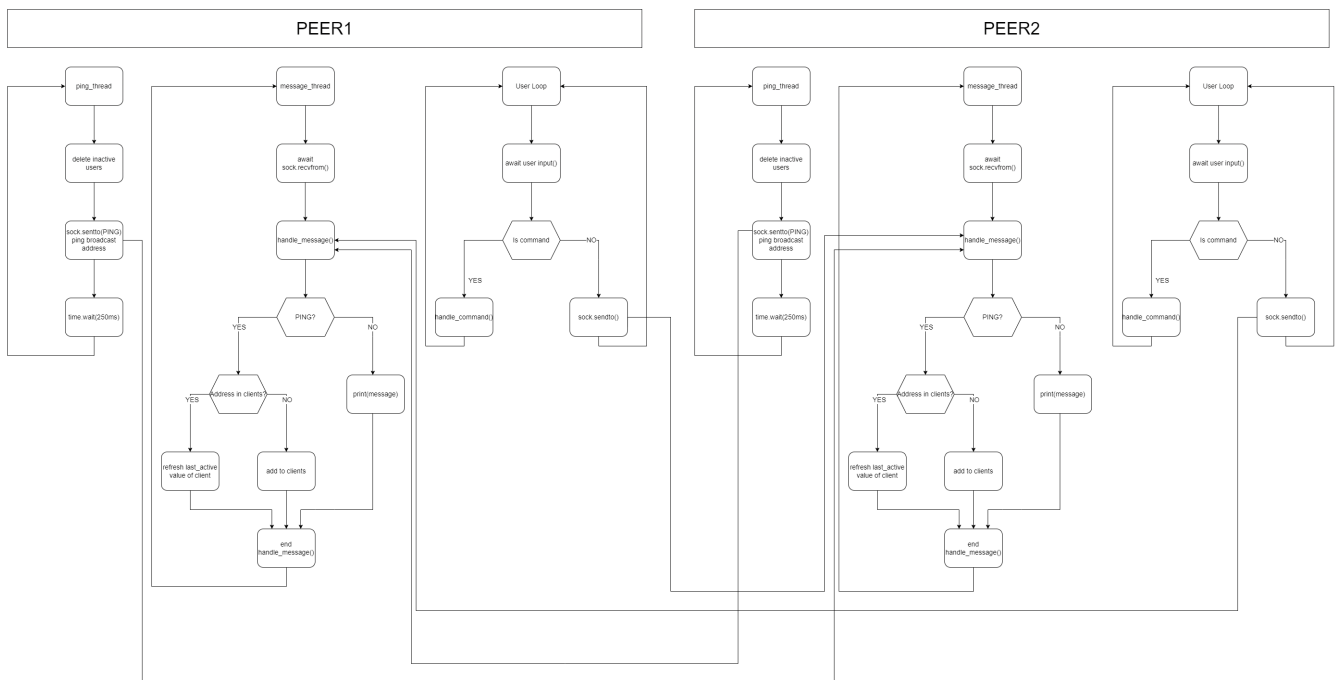
Moduł ui.py odpowiada za warstwę prezentacji, w niej pobierane są dane wejściowe dalej przekazywane do funkcji działających na wątkach zdefiniowanych w peer.py. Wstępnie warstwa prezentacji będzie oparta o terminal i Pythonowe funkcje input() i print(), ale w ramach rozwoju aplikacji jest możliwa zmiana na UI napisane w przeznaczonej do tego bibliotece. Moduł client.py pozwala nam na odseparowanie definicji klasy klienta od reszty kodu. Możliwe jest, że będzie konieczne poszerzenie klasy klienta o atrybuty opcjonalne takie jak nick itd.

POPRAWKA: Okazało się, że na potrzeby naszego projektu zbyt wiele okazało się tworzenie klasy klienta, dlatego o klienci są przechowywani jako słownik o kluczu będącym adresem klienta i wartością będącą czasem wysłania ostatniego pingu.

## 5 Zarys koncepcji implementacji

Client opisujemy dwoma najważniejszymi atrybutami:

- address — adres IP i PORT na którym peer słucha
- lastactive — czas w którym ostatnio został odebrany ping od peera. Przechowywane w celu cyklicznego odświeżania listy klientów.



Flowchart w lepszej jakości przesyłamy jako załącznik w formacie svg. Dla każdego z peerów, w celu budowania listy dostępnych hostów oraz przyjmowania wiadomości działają dwa wątki:

1. wątek pingthread odpowiedzialny za:

- cykliczne pingowanie adresu rozgłoszeniowego z wykorzystaniem protokołu UDP, ping powinien dotrzeć do wszystkich aktywnych peerów w sieci
- usuwanie nieaktywnych peerów z listy klientów z wykorzystaniem atrybutu lastactive

Funkcja tego wątku jest wykonywana co interwał czasowy, wstępnie 250ms. Peer jest uznany za nieaktywny jeśli przez 5s nie wysłał on żadnego pingu.

2. wątek przyjmujący wiadomości(w tym pingu) i aktualizujący listę klientów. Dla klienta niewystępującego na liście, dopisuje go do listy, dla klienta występującego już na liście, aktualizuje wartość atrybutu lastactive. Wątek ten działa w nieskończonej pętli.

3. Dodatkowo potrzebujemy jeszcze jednej nieskończonej pętli, w której użytkownik będzie mógł wpisywać treść wiadomości lub korzystać z innych opcji programu (takich jak !clients, !nickname, !exit). Po otrzymaniu inputu od użytkownika który nie jest funkcyjny, wiadomość zostaje rozesłana do każdego z aktywnych peerów z osobna.

Do realizacji tego wykorzystamy jeden socket UDP wprowadzony w tryb broadcast dla każdego z peerów.

## 6 opis najważniejszych rozwiązań funkcjonalnych

Zasada działania naszej aplikacji została opisana w poprzednim akapicie. Protokół który wybraliśmy do realizacji zadania to UDP. Udało nam się dzięki niemu w pełni zrealizować wszystkie zadane funkcjonalności aplikacji. Samo rozgłaszanie pingów musi zachodzić przez UDP, a że dostaliśmy pełną dowolność w sprawie tego jak wyglądać ma wysyłanie wiadomości, to wiadomości również wysyłamy przez UDP, tym razem jednak nie na adres broadcastowy, a do każdego z dostępnych klientów osobno.

Struktury danych:

- Peer - klasa w której zamknęliśmy implementacje koncepcji z poprzedniego akapitu.
- clients - dostępnych klientów przechowujemy w słowniku aktualizowanym co każde otrzymanie pinga w message thread lub co każde wywołanie funkcji działającej na ping threadzie. Sam słownik ma strukturę (klucz : wartość) -(Adres peera : czas ostatnio otrzymanego pinga). Taka struktura zbierająca i aktualizująca dane o peerach okazała się jak najbardziej wystarczająca dla naszego projektu.

Kluczowe funkcje:

### 1. Klasa Peer:

- \_\_init\_\_ - funkcja w której inicjalizowane oraz w niektórych przypadkach przygotowywana są wszystkie potrzebne atrybuty peera: socket, własny adres, lista klientów oraz definionwane są potrzebne wątki.
- start\_threads i stop\_threads - funkcje do startowania i zatrzymywania pracy wątków.
- send\_message - funkcja z której korzysta UI, służy do wysłania podanej wiadomości po uprzednim przystosowaniu jej do naszej aplikacji - dodaniu nicku oraz czasu wysłania wiadomości. Wiadomość jest wysyłana do wszystkich dostępnych peerów oprócz samego siebie.
- \_broadcast\_ping - funkcja działająca na ping\_thread, usuwa nieaktywnych peerów oraz wysyła pinga na adres broadcast(255.255.255.255), operacja ta wykonuje się co 0.5 sekundy.
- \_listen\_for\_messages - funkcja działająca na message\_thread, oczekuje na przysłania wiadomości, jeśli ją otrzyma wywołuje z nią funkcję \_handle\_message

- `_handle_message` - w razie otrzymania pinga, aktualizuje wartość czasu ostatnio wysłanego pinga przez danego klienta w liście klientów. Jeśli jest to wiadomość, przekazuję ją do UI aby ją wyświetlić - obecnie UI jest w pełni zrobione w terminalu, dlatego w przypadku wiadomości nie będącej pingiem, wywołuje się po prostu funkcja `print()`.

## 7 Interfejs użytkownika

UI ograniczyliśmy do zupełnego minimum, użytkownik po uruchomieniu aplikacji może wysyłać wiadomości lub skorzystać z opcji:

- `!clients` - pokazuje listę dostępnych peerów
- `!name [nickname]` - pozwala zmienić nazwę użytkownika na `[nickname]`
- `!exit` - wyłącza aplikację

Dodatkowo w kluczowych momentach wysyłane są wiadomości do użytkownika:

- Przywitanie użytkownika
- Dołączenie do czatu innych użytkowników
- Opuszczenie czatu przez użytkownika
- Zmiana nazwy użytkownika
- Opuszczanie aplikacji

## 8 Pliki konfiguracyjne i logi

Wykorzystaliśmy bibliotekę `logging`, która pomogła nam debugować klasę `peer`. Obecnie logi nie są pokazywane w terminalu, aby to zmienić, należy zmienić `level` z `INFO` na `DEBUG`. Jedynym plikiem pseudo-konfiguracyjnym jaki musieliśmy użyć jest `constants.py`. Przechowane są tam stałe tak samo jak i wartości które można podjąć konfiguracji takie jak `BUFFERSIZE` czy `PORT` na którym działa aplikacja.

## 9 Użyte technologie i narzędzia

Chat został zrealizowany w Pythonie. System operacyjny Linux Ubuntu 22.04.XX LTS. Użyte biblioteki:

- `logging`
- `socket`
- `threading`
- `time`, `datetime` - łatwe operacje na czasie wysyłania oraz otrzymywania wiadomości

## 10 Testy i wyniki testowania

Po uruchomieniu programu zostaje wyświetlona lista możliwych komend, oraz wypisani użytkownicy, którzy już dołączyli. Po wpisaniu `!clients` zostaną wyświetleni wszyscy dostępni użytkownicy. Niżej widać kilku kolejnych klientów, którzy dołączają do czatu i wyświetlają listę dostępnych użytkowników.

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
!clients
{('172.17.0.2', 5000): 1674129718.1180701}
```

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.2 joined the chat
!clients
{('172.17.0.3', 5000): 1674129917.6929305, ('172.17.0.2', 5000): 1674129917.658035}
```

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.2 joined the chat
Client 172.17.0.3 joined the chat
!clients
{('172.17.0.4', 5000): 1674129972.6010728, ('172.17.0.2', 5000): 1674129972.60615, ('172.17.0.3', 5000): 1674129972.6021023}
```

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.2 joined the chat
Client 172.17.0.4 joined the chat
Client 172.17.0.3 joined the chat
!clients
{('172.17.0.5', 5000): 1674129989.9308925, ('172.17.0.2', 5000): 1674129989.9318054, ('172.17.0.4', 5000): 1674129989.9309165, ('172.17.0.3', 5000): 1674129989.9317737}
```



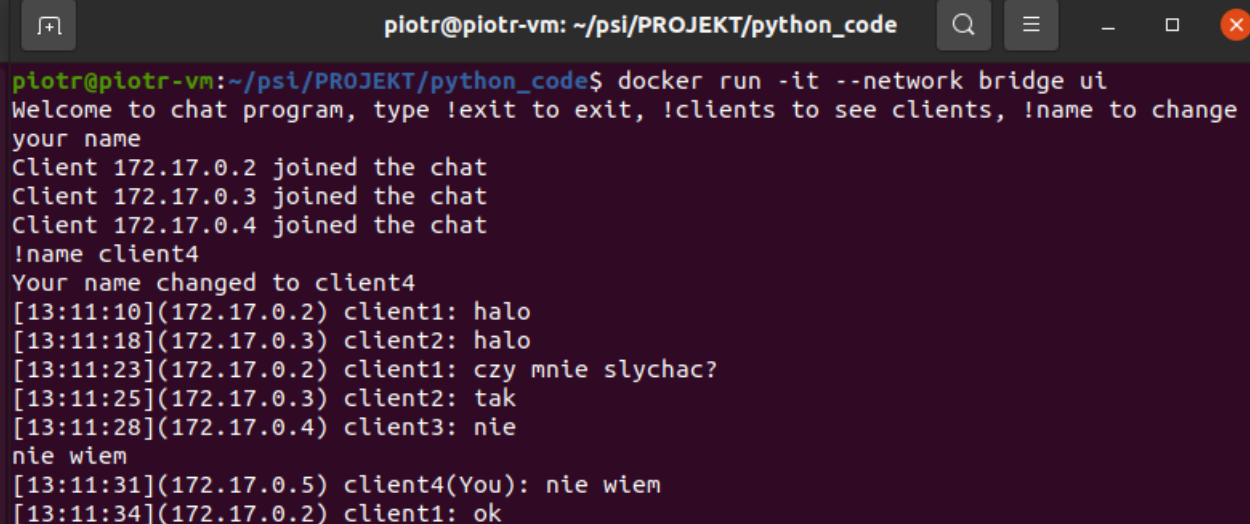
Po zmianie nazwy użytkownika za pomocą `!name`, inni użytkownicy widzą nazwę autora wiadomości. Niżej widać czat z perspektywy kilku klientów.

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.3 joined the chat
Client 172.17.0.4 joined the chat
Client 172.17.0.5 joined the chat
!name client1
Your name changed to client1
halo
[13:11:10](172.17.0.2) client1(You): halo
[13:11:18](172.17.0.3) client2: halo
czy mnie slychac?
[13:11:23](172.17.0.2) client1(You): czy mnie slychac?
[13:11:25](172.17.0.3) client2: tak
[13:11:28](172.17.0.4) client3: nie
[13:11:31](172.17.0.5) client4: nie wiem
ok
[13:11:34](172.17.0.2) client1(You): ok
█

piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.3 joined the chat
Client 172.17.0.2 joined the chat
Client 172.17.0.5 joined the chat
!name client3
Your name changed to client3
[13:11:10](172.17.0.2) client1: halo
[13:11:18](172.17.0.3) client2: halo
[13:11:23](172.17.0.2) client1: czy mnie slychac?
[13:11:25](172.17.0.3) client2: tak
nie
[13:11:28](172.17.0.4) client3(You): nie
[13:11:31](172.17.0.5) client4: nie wiem
[13:11:34](172.17.0.2) client1: ok
█
```

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.2 joined the chat
Client 172.17.0.4 joined the chat
Client 172.17.0.5 joined the chat
!name client2
Your name changed to client2
[13:11:10](172.17.0.2) client1: halo
halo
[13:11:18](172.17.0.3) client2(You): halo
[13:11:23](172.17.0.2) client1: czy mnie slychac?
tak
[13:11:25](172.17.0.3) client2(You): tak
[13:11:28](172.17.0.4) client3: nie
[13:11:31](172.17.0.5) client4: nie wiem
[13:11:34](172.17.0.2) client1: ok

```

A terminal window titled "piotr@piotr-vm: ~/psi/PROJEKT/python\_code" with standard window controls (search, menu, zoom, close). The terminal displays the same chat program output as the first block, but with a different sequence of events: Client 172.17.0.3 joins after Client 172.17.0.2, and Client 172.17.0.4 joins last. The user changes their name to "client4". The chat messages follow a similar pattern, with Client 1 (client1) asking "czy mnie slychac?" and Client 2 (client2) replying "tak". Client 4 (client4) then says "nie wiem".

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.2 joined the chat
Client 172.17.0.3 joined the chat
Client 172.17.0.4 joined the chat
!name client4
Your name changed to client4
[13:11:10](172.17.0.2) client1: halo
[13:11:18](172.17.0.3) client2: halo
[13:11:23](172.17.0.2) client1: czy mnie slychac?
[13:11:25](172.17.0.3) client2: tak
[13:11:28](172.17.0.4) client3: nie
nie wiem
[13:11:31](172.17.0.5) client4(You): nie wiem
[13:11:34](172.17.0.2) client1: ok

```

Kiedy któryś z użytkowników opuści czat, zostanie wyświetlona informacja.

```
piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.3 joined the chat
Client 172.17.0.4 joined the chat
Client 172.17.0.5 joined the chat
!name client1
Your name changed to client1
halo
[13:11:10](172.17.0.2) client1(You): halo
[13:11:18](172.17.0.3) client2: halo
czy mnie slychac?
[13:11:23](172.17.0.2) client1(You): czy mnie slychac?
[13:11:25](172.17.0.3) client2: tak
[13:11:28](172.17.0.4) client3: nie
[13:11:31](172.17.0.5) client4: nie wiem
ok
[13:11:34](172.17.0.2) client1(You): ok
!exit
Exiting

piotr@piotr-vm:~/psi/PROJEKT/python_code$ docker run -it --network bridge ui
Welcome to chat program, type !exit to exit, !clients to see clients, !name to change
your name
Client 172.17.0.3 joined the chat
Client 172.17.0.2 joined the chat
Client 172.17.0.5 joined the chat
!name client3
Your name changed to client3
[13:11:10](172.17.0.2) client1: halo
[13:11:18](172.17.0.3) client2: halo
[13:11:23](172.17.0.2) client1: czy mnie slychac?
[13:11:25](172.17.0.3) client2: tak
nie
[13:11:28](172.17.0.4) client3(You): nie
[13:11:31](172.17.0.5) client4: nie wiem
[13:11:34](172.17.0.2) client1: ok
Client 172.17.0.2 left the chat
```