Arxen

Aplikacja do rozmów konferencyjnych

Grzegorz Kuliński

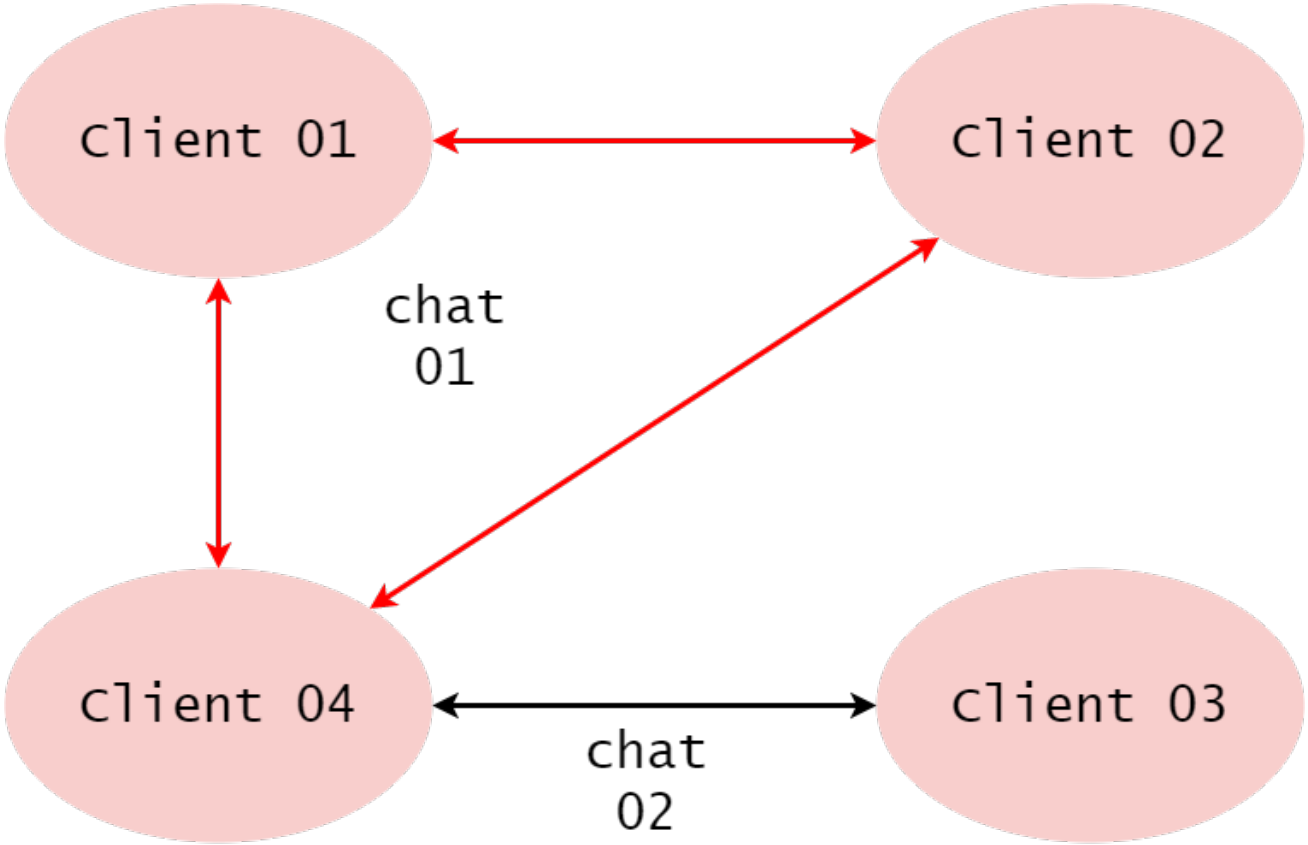
Overview

O czym będzie ta demonstracja

Co robi nasza aplikacja?

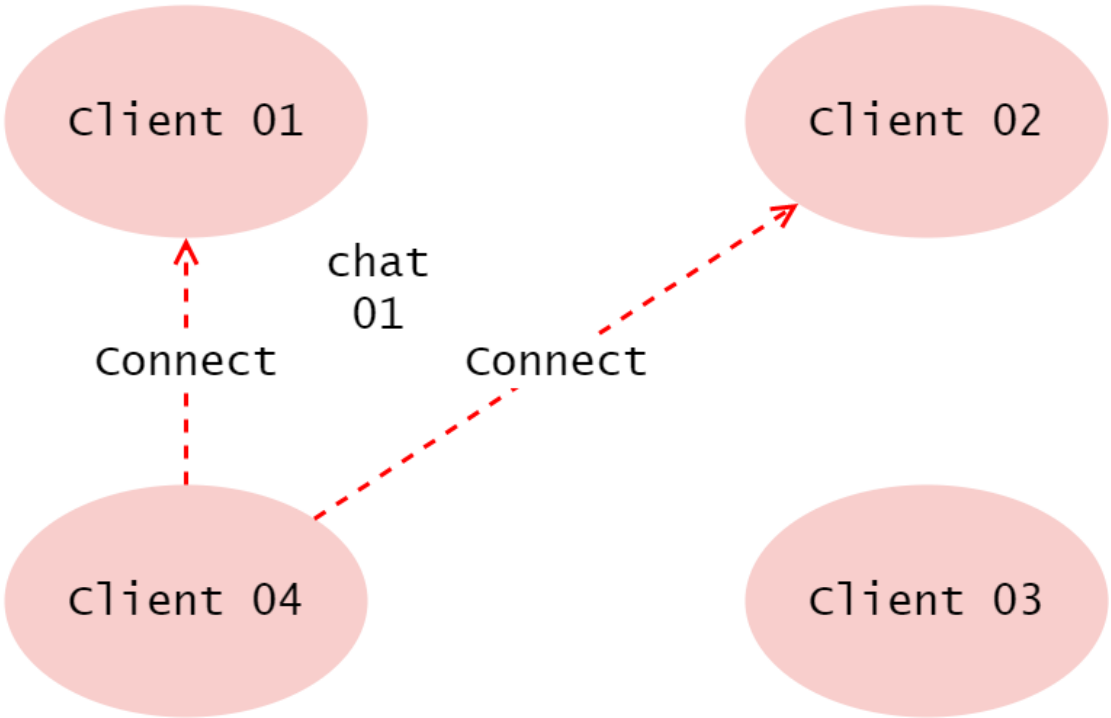
Arxen

Aplikacja do komunikacji p2p między użytkownikami bez udział serwera.



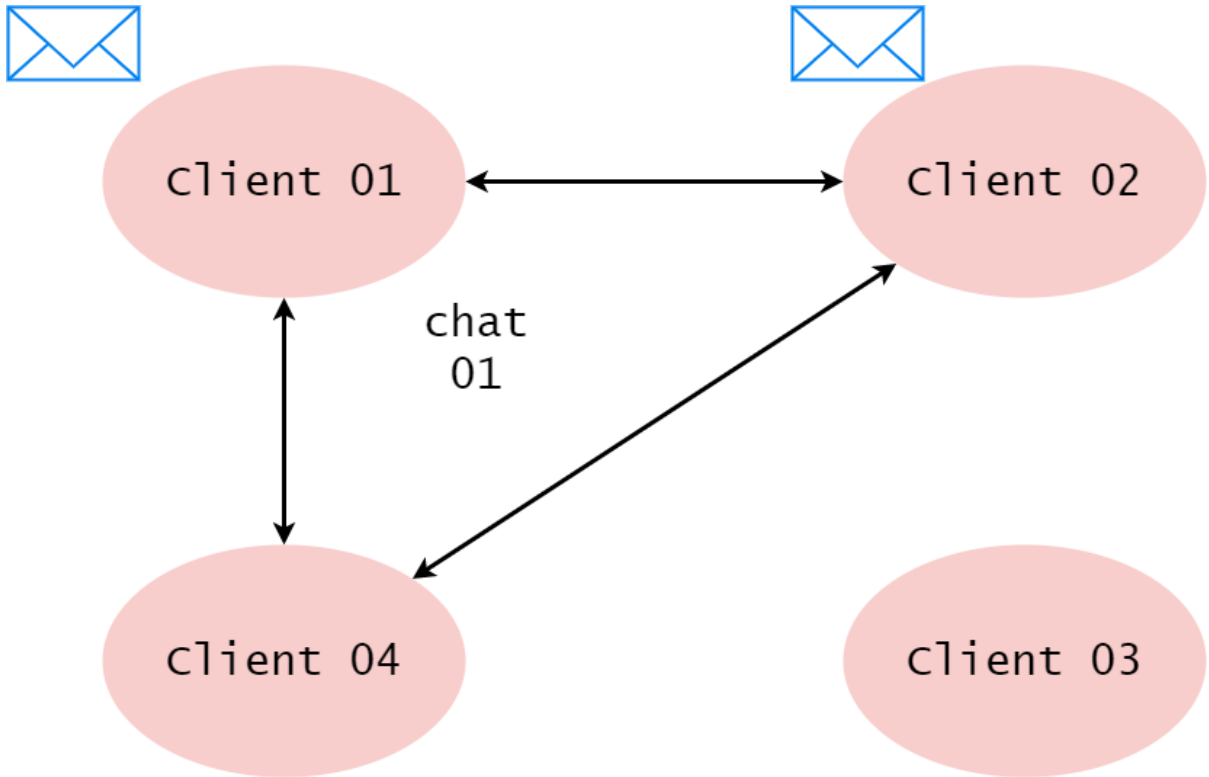
Problemy

1. Jak odnaleźć IP docelowego klienta bez udziału serwera?



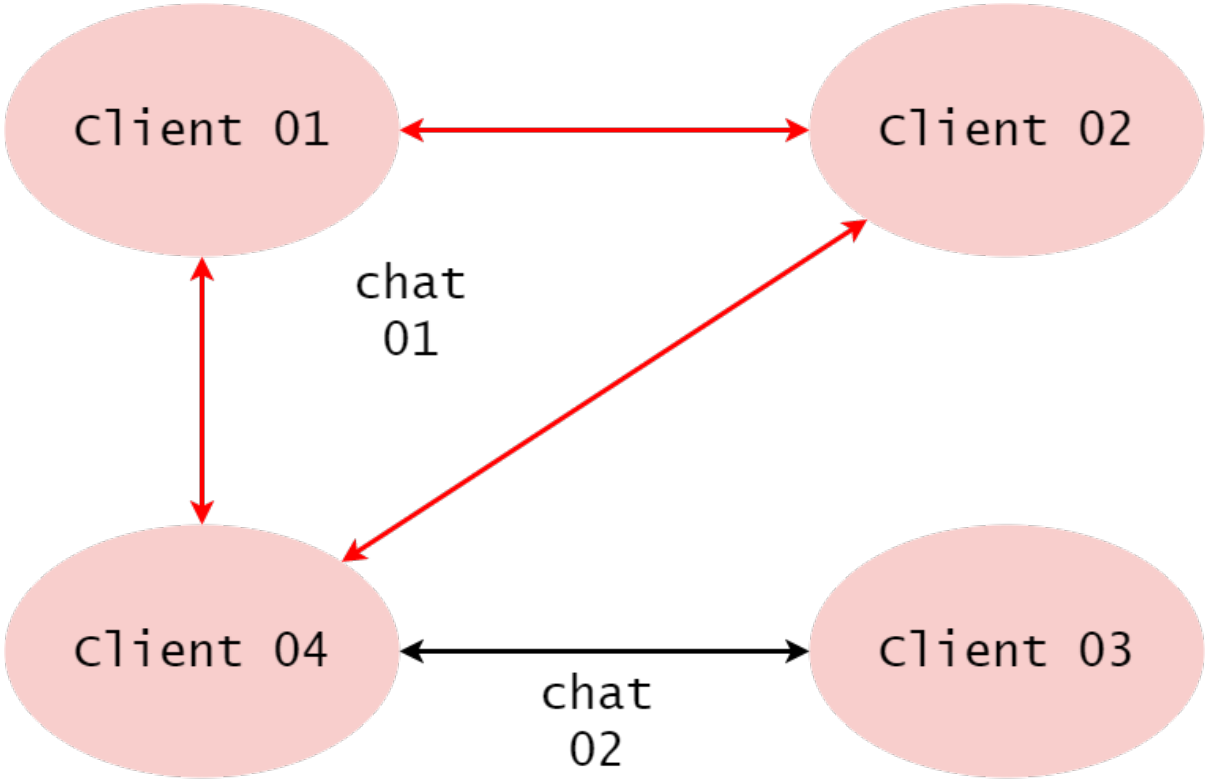
Problemy

1. Jak odnaleźć IP docelowego klienta bez udziału serwera?
2. Kto zarządza wiadomościami w czacie? Kto je zapisuje? Co jeśli nowy użytkownik dołączy do istniejącego czatu?



Problemy

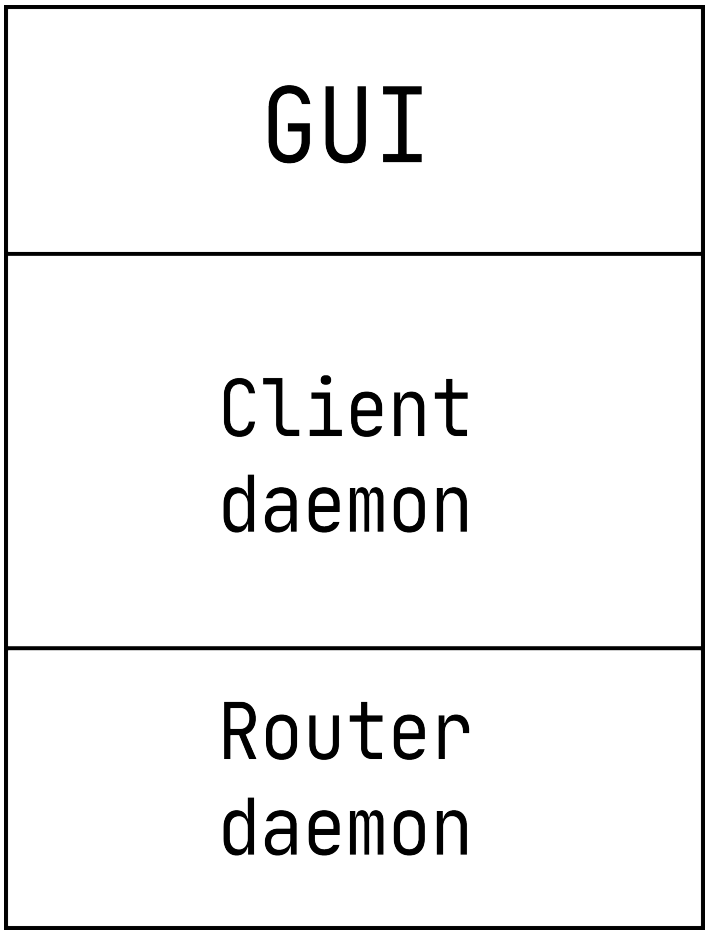
1. Jak odnaleźć IP docelowego klienta bez udziału serwera?
2. Kto zarządza wiadomościami w czacie? Kto je zapisuje? Co jeśli nowy użytkownik dołączy do istniejącego czatu?
3. Co jeśli użytkownik chce korzystać ze swojego konta na kilku urządzeniach?



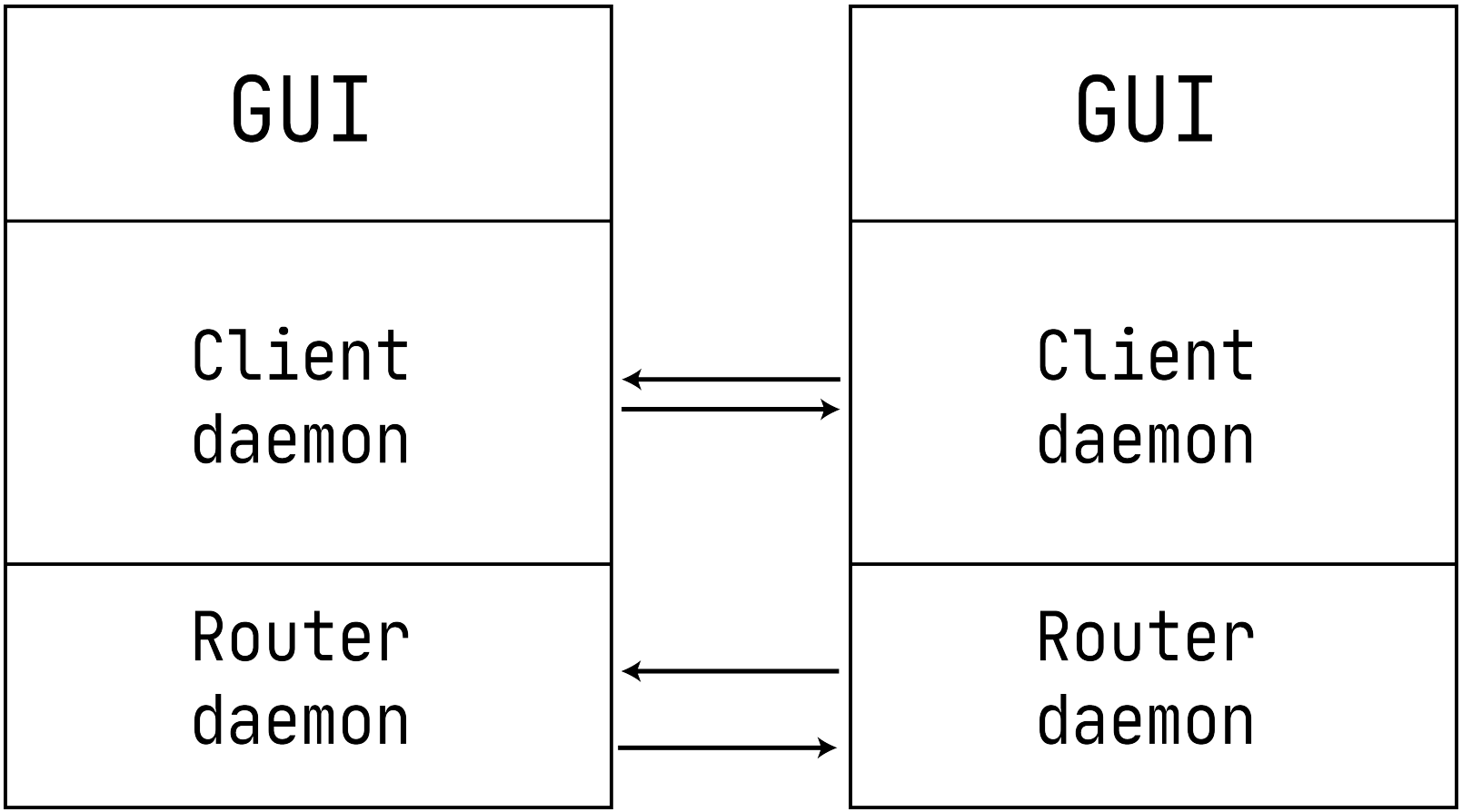
Architektura

Architektura warstwowa

Podział na warstwy

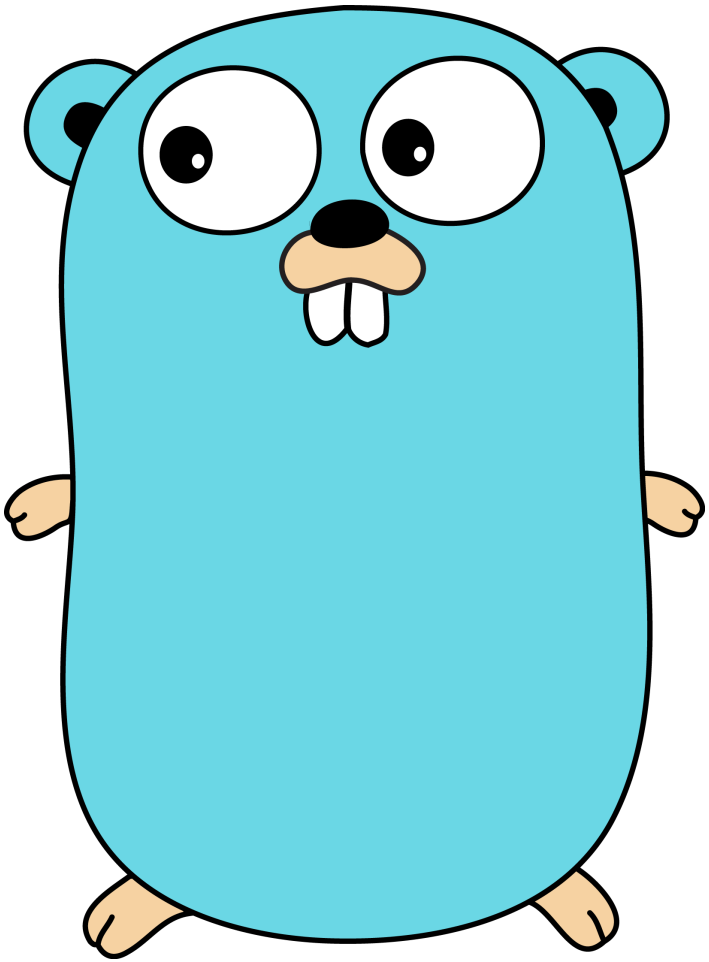


Architektura warstwowa (2)



Technologie

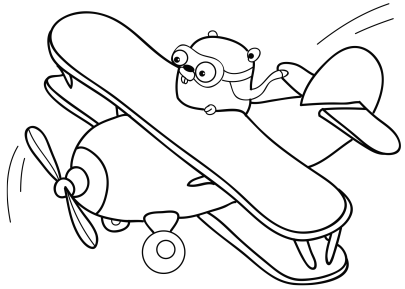
Go



AGH

Go

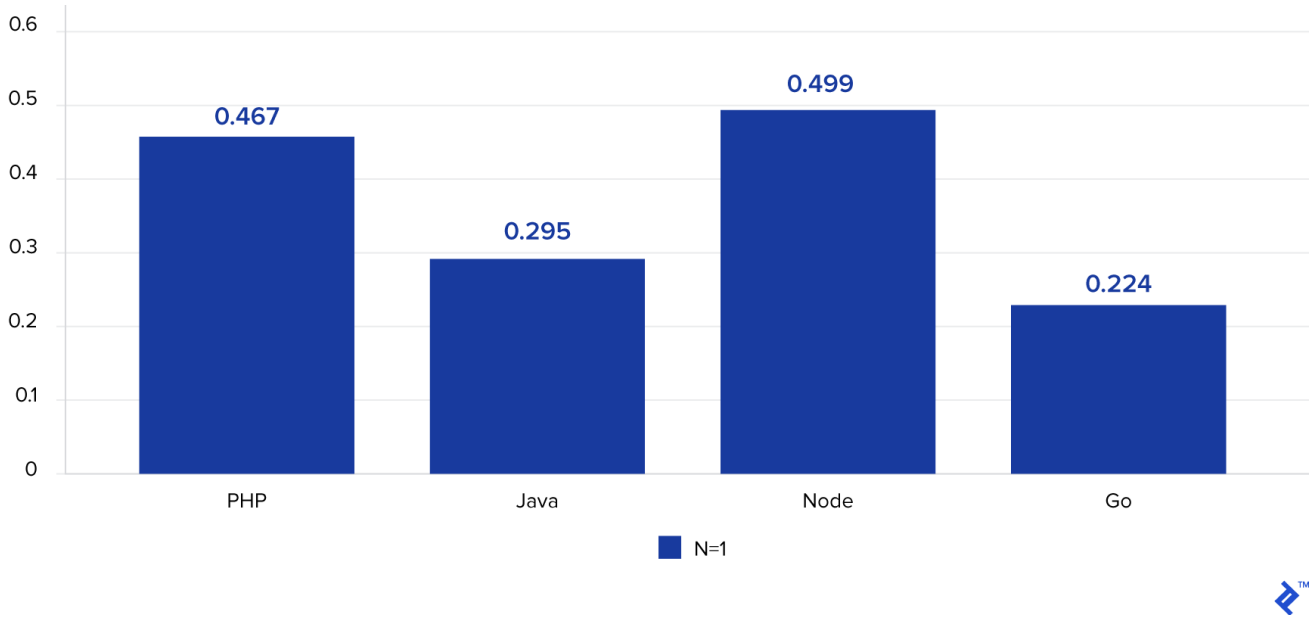
1. cross-platform



AGH

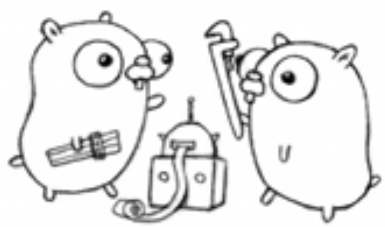
Go

1. cross-platform
2. Wydajny



Go

1. cross-platform
2. Wydajny
3. Wbudowany model wielowątkowości

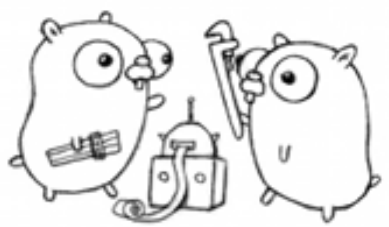


Example

```
func say(s string) {
        for i := 0; i < 5; i++ {
                time.Sleep(100 *
                    time.Millisecond)
                fmt.Println(s)
        }
}

func main() {
        go say("world")
        say("hello")
}
```

AGH

Wielowątkowość w Go



Channel: typ wbudowany pozwalający na wymiane komunikatów między goroutines operator <- oraz ->

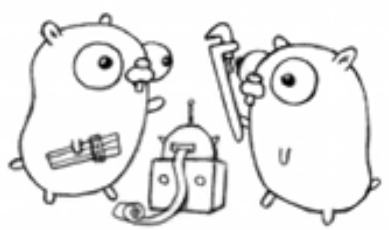
Goroutine: Wbudowany w język system zarządzania wątkami

AGH

Wielowątkowość w Go (2)

przykład channelu

```
func sum(s []int, c chan int) {
        sum := 0
        for _, v := range s {
                sum += v
        }
        c <- sum // send sum to c
}

func main() {
        s := []int{7, 2, 8, -9, 4, 0}

        c := make(chan int)
        go sum(s[:len(s)/2], c)
        go sum(s[len(s)/2:], c)
        x, y := <-c, <-c // receive from c

        fmt.Println(x, y, x+y)
}
```



AGH

RSocket

Protokół warstwy aplikacji oparty na TCP lub WebSocket pozwalający na loadbalancing

AGH

RSocket

Wspierane tryby komunikacji:

- request/response (stream of 1)
- request/stream (finite stream of many)
- fire-and-forget (no response)
- channel (bi-directional streams)

RSocket

Przykład prostego serwera obsługującego Request/Response

Example

```
func main() {
        err := rsocket.Receive().
                Resume().
                Fragment(1024).
                Acceptor(func(setup payload.SetupPayload,
                        sendingSocket rsocket.CloseableRSocket)
                        (rsocket.RSocket, error) {
                        // bind responder
                        return rsocket.NewAbstractSocket(
                                rsocket.RequestResponse(func(msg payload.Payload)
                                    mono.Mono {
                                        return mono.Just(msg)
                                }),
                        ), nil
                }).
                Transport("tcp://127.0.0.1:7878").
                Serve(context.Background())
        panic(err)
}
```

GraphQl

Alternatywa dla RESTful APIs.

definicja

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

zapytanie

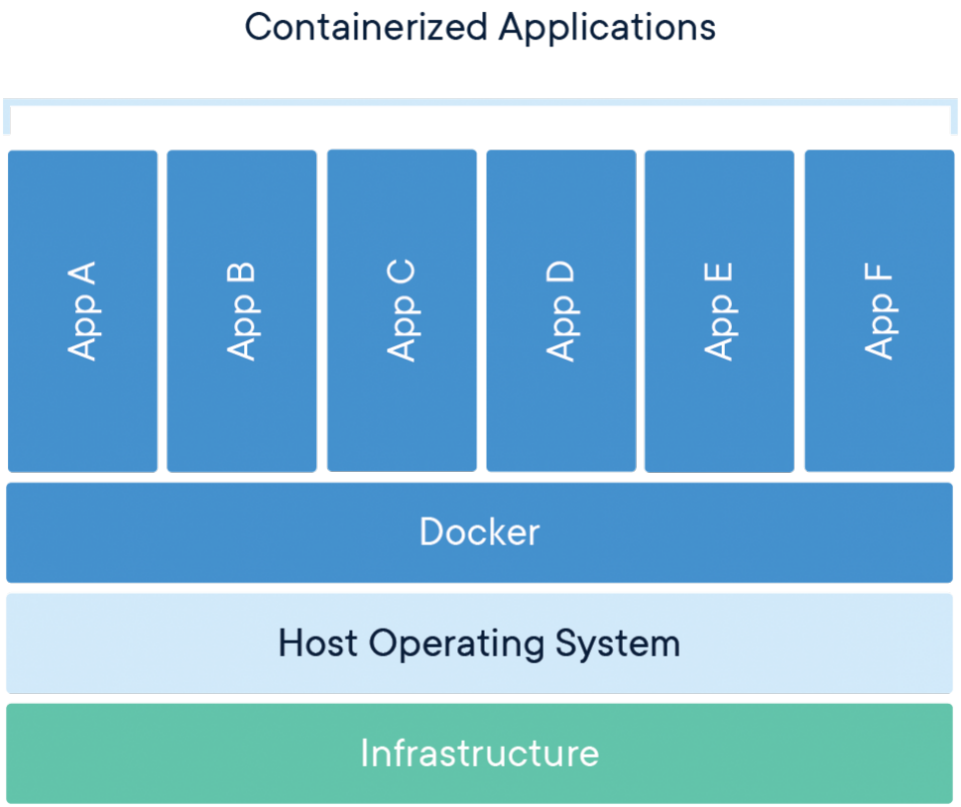
```
{
  me {
    name
  }
}

{
  "me": {
    "name": "Luke Skywalker"
  }
}
```

odpowiedź

```
{
  "me": {
    "name": "Luke Skywalker"
  }
}
```

Docker

Metoda konteneryzacji oprogramowania wspierająca deweloperów w codziennych zadaniach.



Instrukcje

- ADD
- COPY
- ENV
- EXPOSE
- FROM
- LABEL
- STOPSIGNAL
- USER
- VOLUME
- WORKDIR

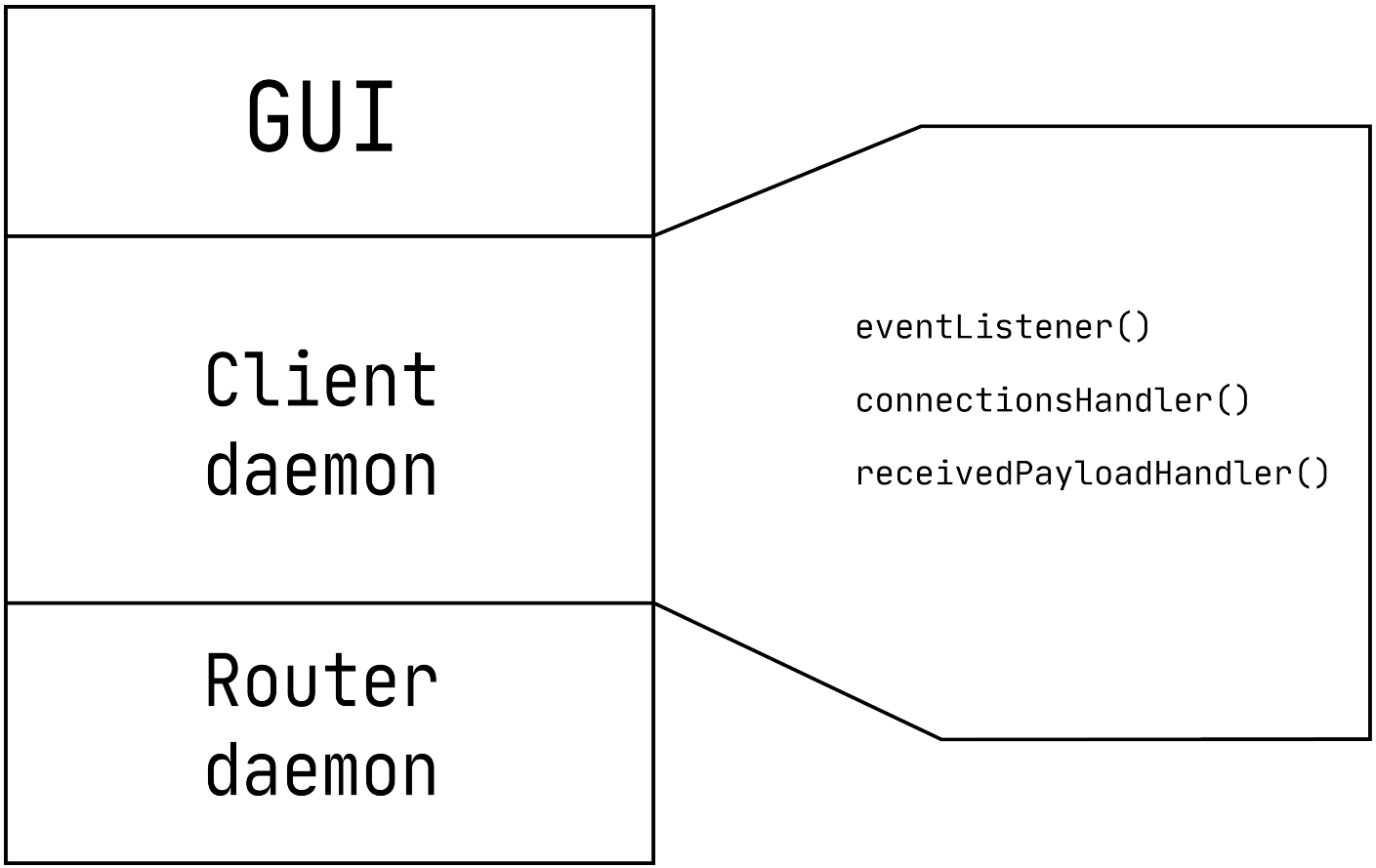
Dockerfile

```
FROM golang:1.14.0-alpine

ENV GO111MODULE=on
ENV PORT=8000

WORKDIR /app

ADD . /app

ADD go.mod .
ADD go.sum .
RUN go mod download

# build command

RUN go build -o /app/main.out .

EXPOSE 80
EXPOSE 8000
EXPOSE 7878
EXPOSE 7879
EXPOSE 8085

CMD ["./main.out"]
```

Client daemon

Struktura warstwy Client daemon

Główne elementy client daemon

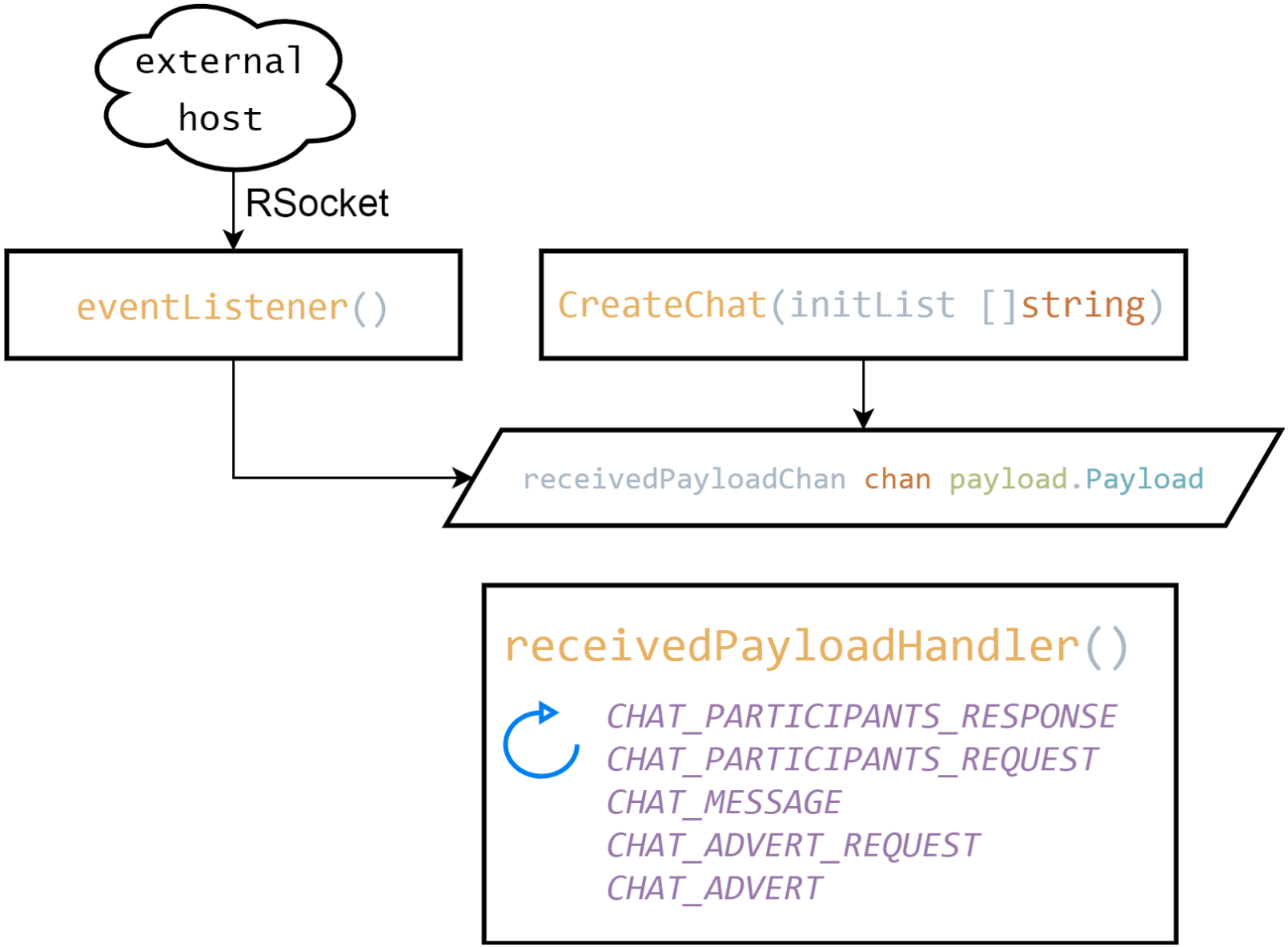


Client daemon

Client - wybrane elementy

```
// Client: basic struct handling connections between other clients
type Client struct {
        userIP     string
        clientsIPs map[string]bool // clientIP : status

        chatList             map[string]*chat.Chat           // chatID, *Chat

        sendDataList         map[string]chan payload.Payload
        // payload and target chat format: map[clientIP] payload(message, chatID)

        receivedPayloadChan chan payload.Payload
        // channel with all incoming payloads

        FriendsList map[string]*gql.Friend // map[friendsNick]Friend

}
```

Client daemon - Struktura logiczna



AGH

Client daemon

Proces tworzenia nowego chatu