SSE

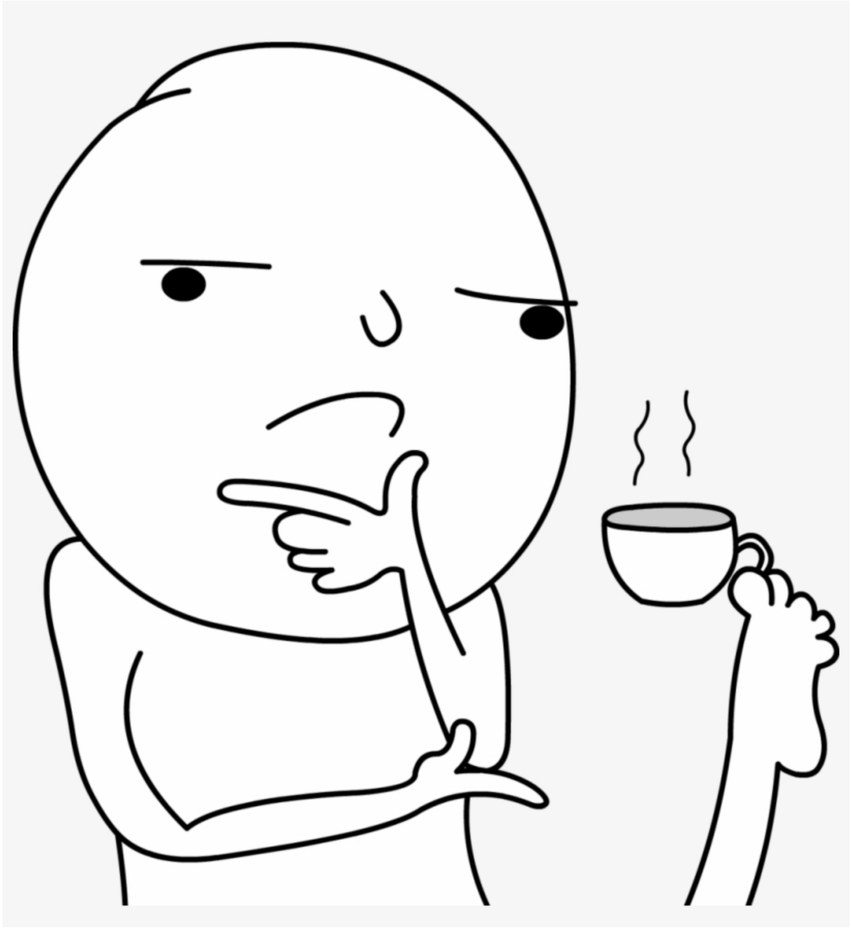
Server-Sent events (Event Streams)

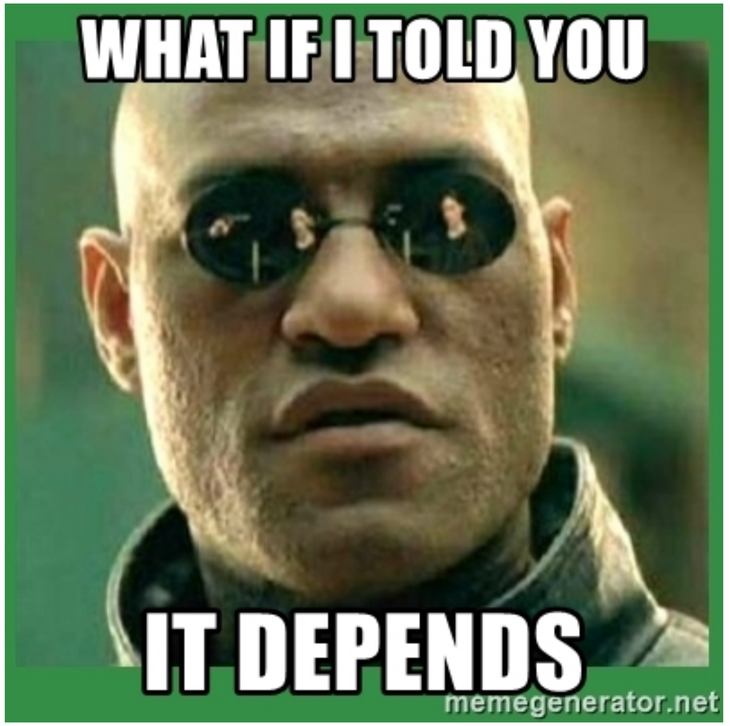
Jadwiga Słowik, 13.05.2022

Problem

Problem

Jak efektywnie zaimplementować komunikację pomiędzy klientem a serwerem?




WHAT IF I TOLD YOU
IT DEPENDS
memegenerator.net

Zależy od konkretnego kontekstu

Zależy od konkretnego kontekstu

- Jaki jest kierunek komunikacji
  - Serwer -> Klient
  - Klient -> Serwer
  - Obustronna?

Zależy od konkretnego kontekstu

- Jaki jest kierunek komunikacji
  - Serwer -> Klient
  - Klient -> Serwer
  - Obustronna?
- Jak często występuje aktualizacja danych?

Zależy od konkretnego kontekstu

- Jaki jest kierunek komunikacji
  - Serwer -> Klient
  - Klient -> Serwer
  - Obustronna?
- Jak często występuje aktualizacja danych?
- Jakie jest obciążenie serwera? (Np. Liczba klientów / równoczesnych zapytań)

Zależy od konkretnego kontekstu

- Jaki jest kierunek komunikacji
  - Serwer -> Klient
  - Klient -> Serwer
  - Obustronna?
- Jak często występuje aktualizacja danych?
- Jakie jest obciążenie serwera? (Np. Liczba klientów / równoczesnych zapytań)
- Jakie są technikalia klientów? (Np. Czy są to komponenty IoT low-end, mobile, web??)

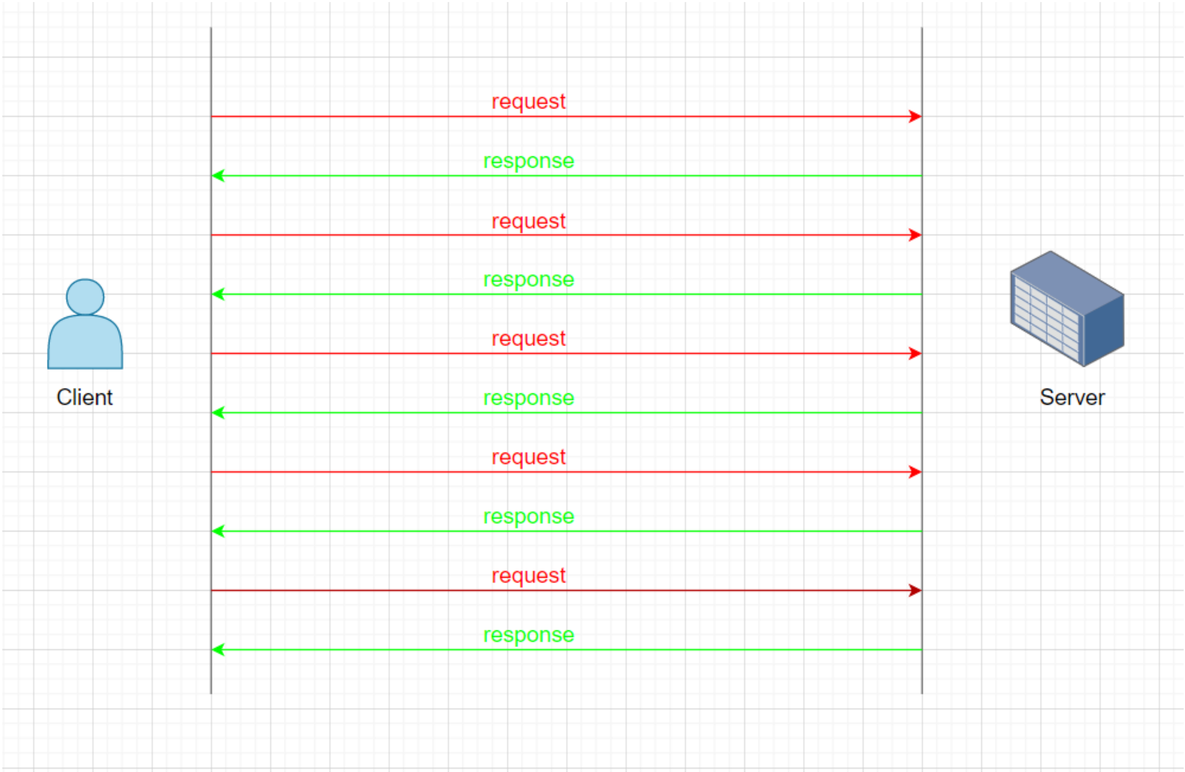
Zależy od konkretnego kontekstu

- Jaki jest kierunek komunikacji
  - Serwer -> Klient
  - Klient -> Serwer
  - Obustronna?
- Jak często występuje aktualizacja danych?
- Jakie jest obciążenie serwera? (Np. Liczba klientów / równoczesnych zapytań)
- Jakie są technikalia klientów? (Np. Czy są to komponenty IoT low-end, mobile, web??)

Potencjalne rozwiązania

Potencjalne rozwiązania

Http Polling / Short polling



- okresowo wysyłamy zapytanie REST do serwera z zapytaniem o gotowe dane

Potencjalne rozwiązania

Http Polling / Short polling

- **Zalety**
  - Prosta implementacja

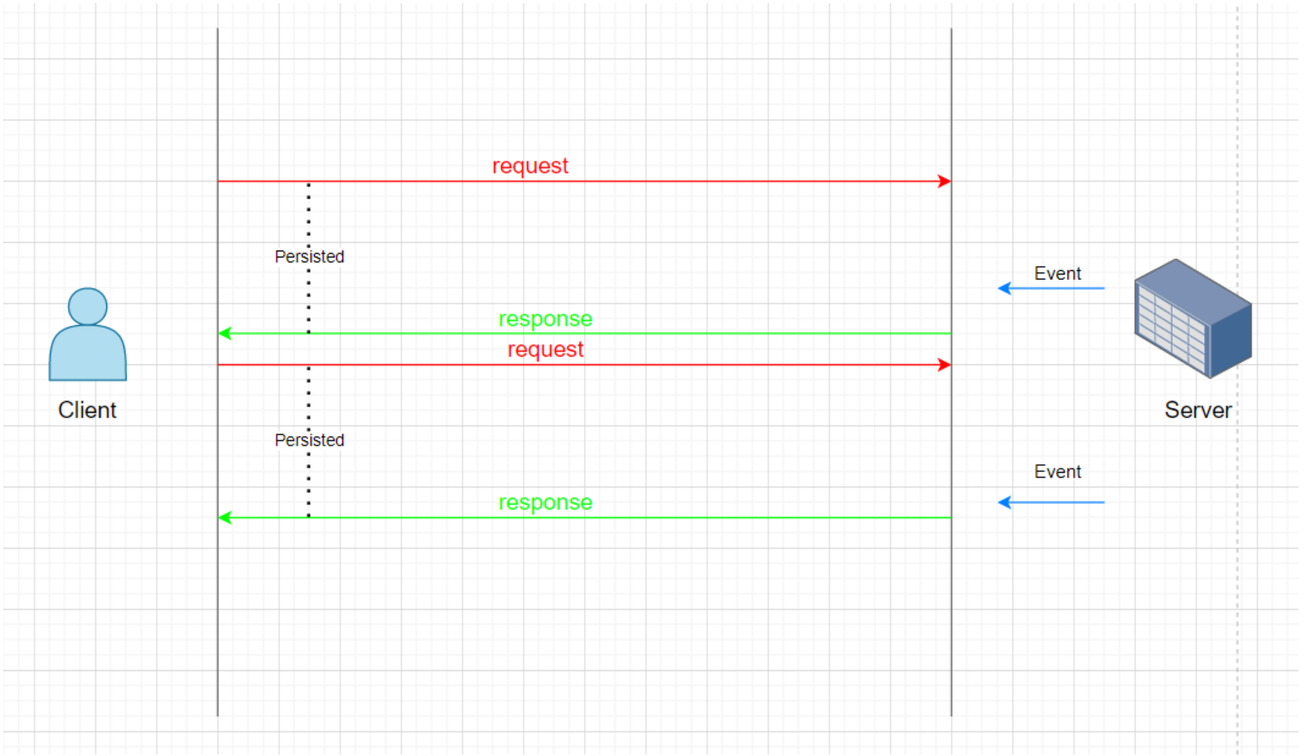
Potencjalne rozwiązania

Http Polling / Short polling

- **Zalety**
  - Prosta implementacja
- **Wady**
  - Duże obciążenie serwera zapytaniami
  - Duże obciążenie procesora i baterii telefonu
  - Nie otrzymujemy nowych danych natychmiast, tylko w momencie wysyłania zapytania przez klienta (“udawany streaming”, dyskretyzacja strumienia)

Potencjalne rozwiązania

Http Long Polling



- Klient wysyła pojedyncze zapytanie i w momencie otrzymania odpowiedzi, wysyła kolejne
- Serwer utrzymuje otwarte połączenie HTTP, dopóki nie pojawią się nowe dane bądź nastąpi timeout.

Potencjalne rozwiązania

Http long polling

- **Zalety**
  - Redukcja liczby niepotrzebnych zapytań w porównaniu do poprzedniego rozwiązania

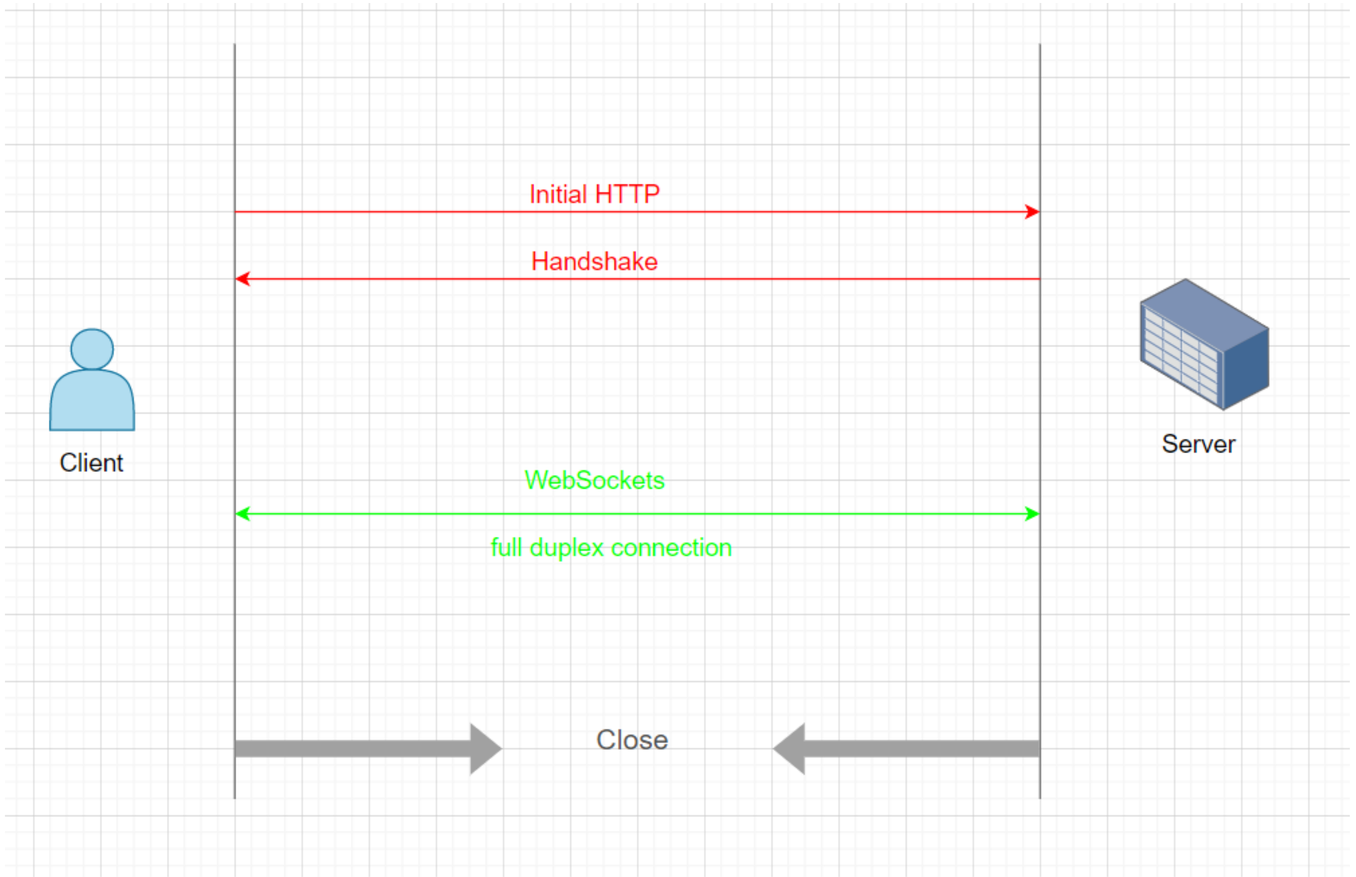
Potencjalne rozwiązania

Http long polling

- **Zalety**
  - Redukcja liczby niepotrzebnych zapytań w porównaniu do poprzedniego rozwiązania
- **Wady**
  - Dodatkowy narzut pracy serwera związany z utrzymywaniem otwartych połączeń z klientami
  - Problemy ze skalowalnością (np. klient ma kilka otwartych połączeń o ten sam zasób)

Potencjalne rozwiązania

Web Sockets



- Dwustronne połączenie pomiędzy klientem a serwerem (full-duplex, stateful)

Potencjalne rozwiązania

Web Sockets

- **Zalety**
  - Rzeczywiście implementuje real-time streaming, a nie tylko naśladuje
  - Mniej zapytań, mniejsze zużycie zasobów
  - Nie ma potrzeby specjalnej obsługi otwartych zapytań przez serwer

Potencjalne rozwiązania

Web Sockets

- **Zalety**
  - Rzeczywiście implementuje real-time streaming, a nie tylko naśladuje
  - Mniej zapytań, mniejsze zużycie zasobów
  - Nie ma potrzeby specjalnej obsługi otwartych zapytań przez serwer
- **Wady**
  - Inny protokół, dodatkowe biblioteki
  - Czy to nie jest overkill?

Server-sent events (SSE)

Server-sent events (SSE)

- **Jednostronne** połączenie pomiędzy serwerem a klientem

Server-sent events

- **Jednostronne** połączenie pomiędzy serwerem a klientem
- **Lekki** protokół korzystający z HTTP Streaming

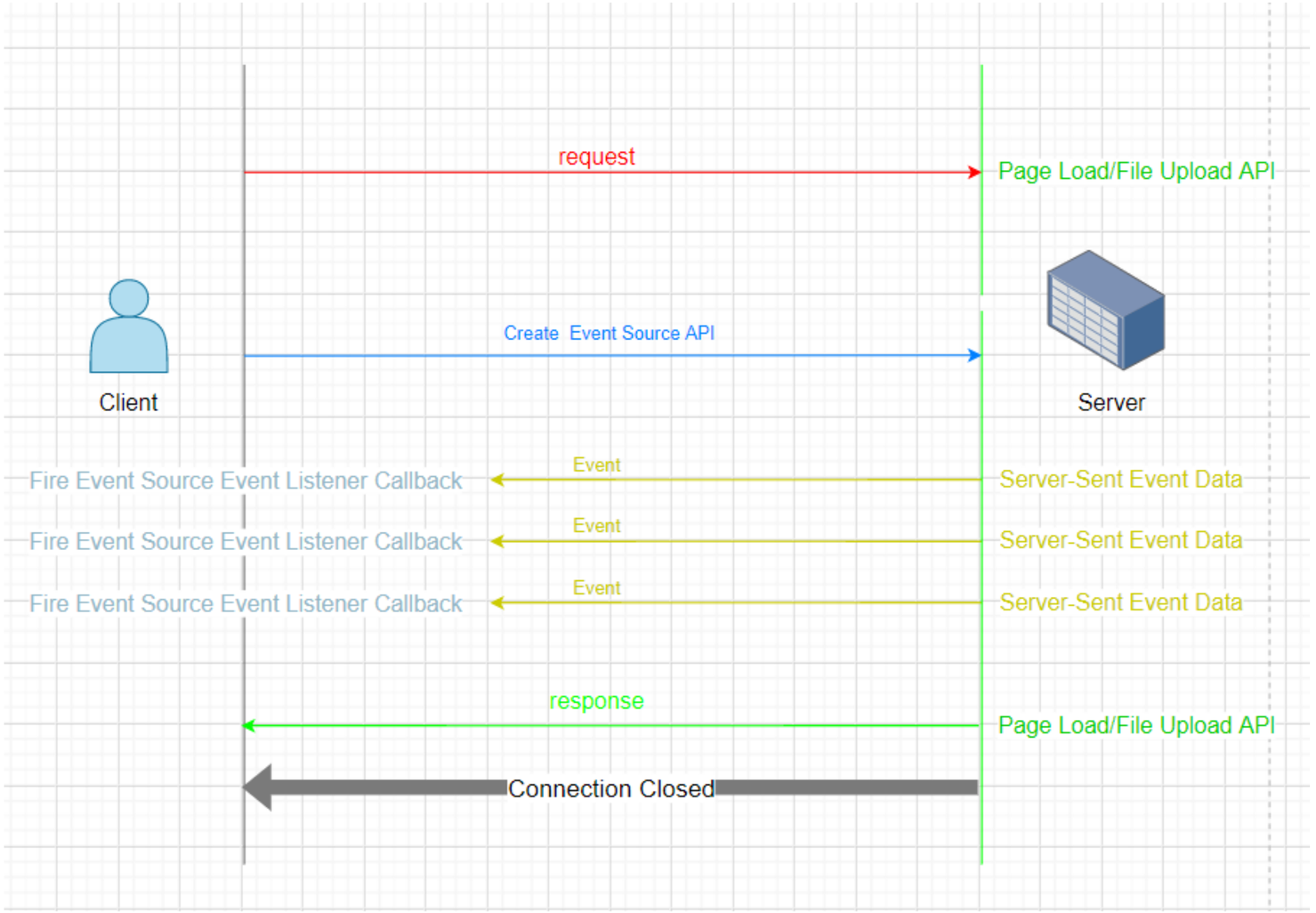
Server-sent events

- **Jednostronne** połączenie pomiędzy serwerem a klientem
- **Lekki** protokół korzystający z HTTP Streaming
- Server-sent Event Source API jest **częścią HTML5**

Server-sent events

- **Jednostronne** połączenie pomiędzy serwerem a klientem
- **Lekki** protokół korzystający z HTTP Streaming
- Server-sent Event Source API jest **częścią HTML5**
- Wydajna pamięciowo implementacja XHR streaming

Server-sent events



Streaming from the REST API

Firebase REST endpoints support the EventSource / Server-Sent Events protocol. To stream changes to a single location in your Realtime Database, you need to do a few things:

- Set the client's Accept header to `"text/event-stream"`
- Respect HTTP Redirects, in particular HTTP status code 307
- If the location requires permission to read, you must include the `auth` parameter

In return, the server will send named events as the state of the data at the requested URL changes. The structure of these messages conforms to the `EventSource` protocol.

```
event: event name
data: JSON encoded data payload
```

Server-sent events

- **Zalety**
  - Łatwy w implementacji i prostszy/lżejszy od web sockets
    - Dla Androida: OkHttp-sse
    - Java (Server): SseEmitter
  - Dobra alternatywa dla Web Sockets, gdy nie jest potrzebna obustronna komunikacja
  - Automatyczne wznowienie połączenia (automatic reconnection)

Server-sent events

- **Zalety**
  - Łatwy w implementacji i prostszy/lżejszy od web sockets
    - Dla Androida: OkHttp-sse
    - Java (Server): SseEmitter
  - Dobra alternatywa dla Web Sockets, gdy nie jest potrzebna obustronna komunikacja
  - Automatyczne wznowienie połączenia (automatic reconnection)
- **Wady**
  - Wiadomości SSE są tekstowe (stąd mniej efektywna obsługa wiadomości binarnych)
  - Jednostronna komunikacja
  - Limit na liczbę otwartych połączeń (6 połączeń na przeglądarkę) dla HTTP/1.1 (dla HTTP/2: ok. 100)

Web Sockets vs. SSE

Podział zastosowań

Web Sockets vs. SSE

Podział zastosowań

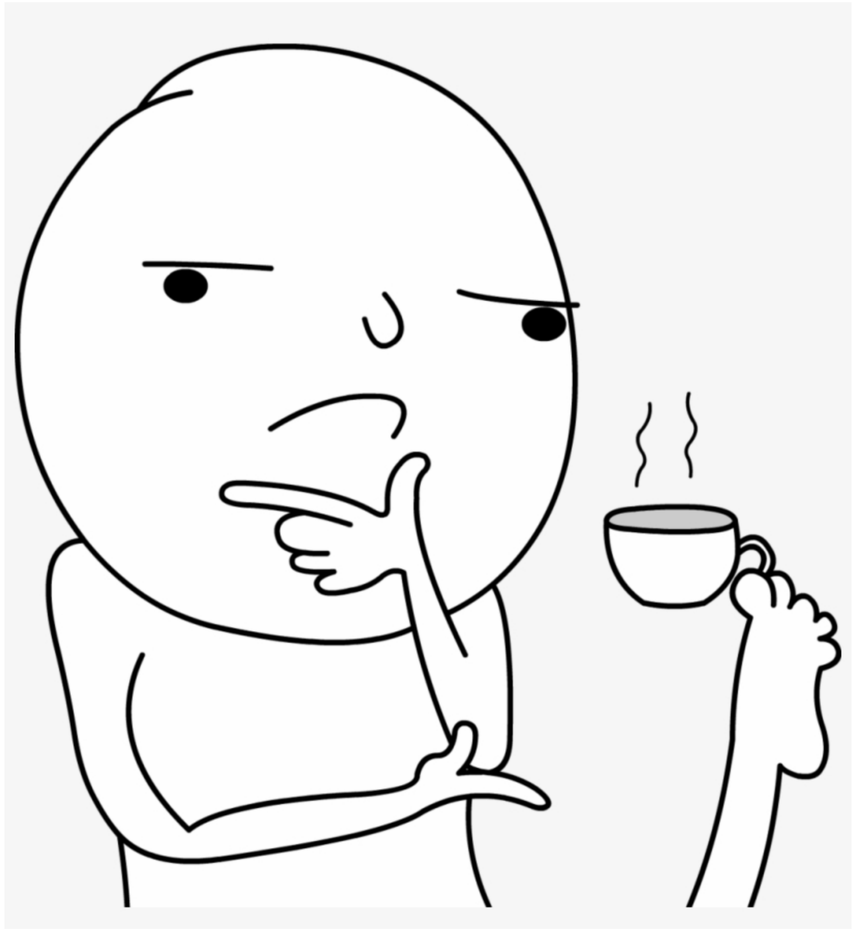
Web Sockets	SSE
Wymagające dwustronnej komunikacji	Wystarczająca jest jednostronna komunikacja
Implementacja chatu	Streamowanie cen, kursów walut/akcji, newsy
Gry w czasie rzeczywistym	Informacje o stanie obliczeń / przetwarzania na serwerze (progress)
Dane binarne	Dane tekstowe
Duża liczba otwartych połączeń z jednym klientem	Mała liczba otwartych połączeń z jednym klientem (dla HTTP/1.1) (<= 6)

Web Sockets vs. SSE

Podział zastosowań

Web Sockets	SSE
Wymagające dwustronnej komunikacji	Wystarczająca jest jednostronna komunikacja
Implementacja chatu	Streamowanie cen, kursów walut/akcji, newsy
Gry w czasie rzeczywistym	Informacje o stanie obliczeń / przetwarzania na serwerze (progress)
Dane binarne	Dane tekstowe
Duża liczba otwartych połączeń z jednym klientem	Mała liczba otwartych połączeń z jednym klientem (dla HTTP/1.1) (<= 6)

Jak zaimplementować?



Jak zaimplementować?

- Odpowiednia struktura wiadomości

Jak zaimplementować?

- Odpowiednia struktura wiadomości
  - **data** - dane w postaci stringa
  - **event** - jeśli mamy kilka różnych zdarzeń, to możemy wybrać, na jakie chcemy się zasubskrybować
  - **id** - id zdarzenia
  - **retry** - liczba określająca czas ponownego połączenia podczas próby wysłania zdarzenia

Jak zaimplementować?

- Odpowiednia struktura wiadomości
- Ustawienie nagłówków

Jak zaimplementować?

- Odpowiednia struktura wiadomości
- Ustawienie nagłówków
  - ‘Content-Type’: ’text/event-stream’
  - ‘Cache-control’: ‘no-cache’
  - ‘Connection’: ‘keep-alive’

Jak zaimplementować?

- Odpowiednia struktura wiadomości
- Ustawienie nagłówków
- Ustanowienie połączenia: klient -> serwer

Jak zaimplementować?

- Odpowiednia struktura wiadomości
- Ustawienie nagłówków
- Ustanowienie połączenia: klient -> serwer
- Nasłuchiwanie po stronie klienta
- Nadawanie po stronie serwera

Jak zaimplementować?

- Odpowiednia struktura wiadomości
- Ustawienie nagłówków
- Ustanowienie połączenia: klient -> serwer
- Nasłuchiwanie po stronie klienta
- Nadawanie po stronie serwera


TALK IS CHEAP
SHOW ME THE CODE
memegenerator.net

Nadawanie po stronie serwera

SseEmitter

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping("/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping("/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping(“/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping(“/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping("/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping(“/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

SseEmitter.SseEventBuilder

`public static interface` **`SseEmitter.SseEventBuilder`**

A builder for an SSE event.

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
`Set<ResponseBodyEmitter.DataWithMediaType>`	**`build`**`()` Return one or more Object-MediaType pairs to write via **`SseEmitter.send(Object, MediaType)`**.
`SseEmitter.SseEventBuilder`	**`comment`**`(String comment)` Add an SSE "comment" line.
`SseEmitter.SseEventBuilder`	**`data`**`(Object object)` Add an SSE "data" line.
`SseEmitter.SseEventBuilder`	**`data`**`(Object object, MediaType mediaType)` Add an SSE "data" line.
`SseEmitter.SseEventBuilder`	**`id`**`(String id)` Add an SSE "id" line.
`SseEmitter.SseEventBuilder`	**`name`**`(String eventName)` Add an SSE "event" line.
`SseEmitter.SseEventBuilder`	**`reconnectTime`**`(long reconnectTimeMillis)` Add an SSE "retry" line.

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping("/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

Nadawanie po stronie serwera

SseEmitter

```
@RequestMapping("/doSth/{count}")
SseEmitter doSth(@PathVariable int count) {
    SseEmitter sseEmitter = new SseEmitter();
    doer
        .doSth(count)
        .subscribe(
            value -> notifyProgress(value, sseEmitter),
            sseEmitter::completeWithError,
            sseEmitter::complete
        );
    return sseEmitter;
}

private void notifyProgress(int value, SseEmitter sseEmitter) {
    try {
        sseEmitter.send(value);
    } catch (IOException e){
        throw new RuntimeException(e);
    }
}
```

Odbieranie po stronie klienta

Android

- Jest wygodna biblioteka - OkHttp3-sse
  - EventSourceListener
  - EventSources