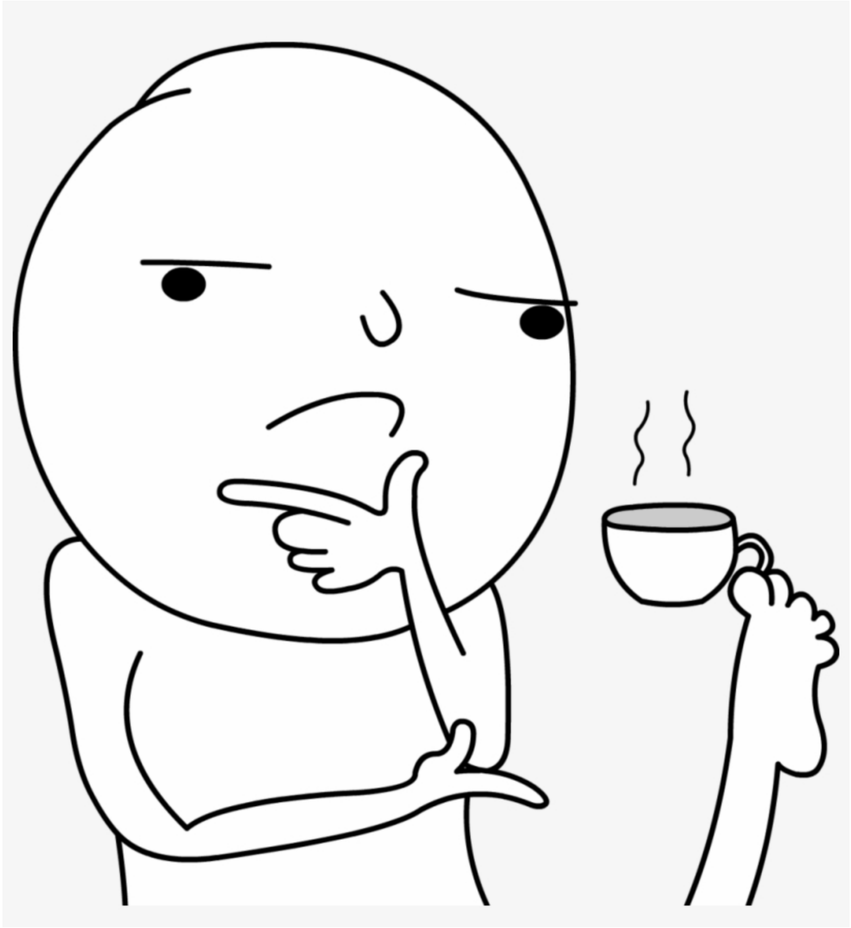
SSE

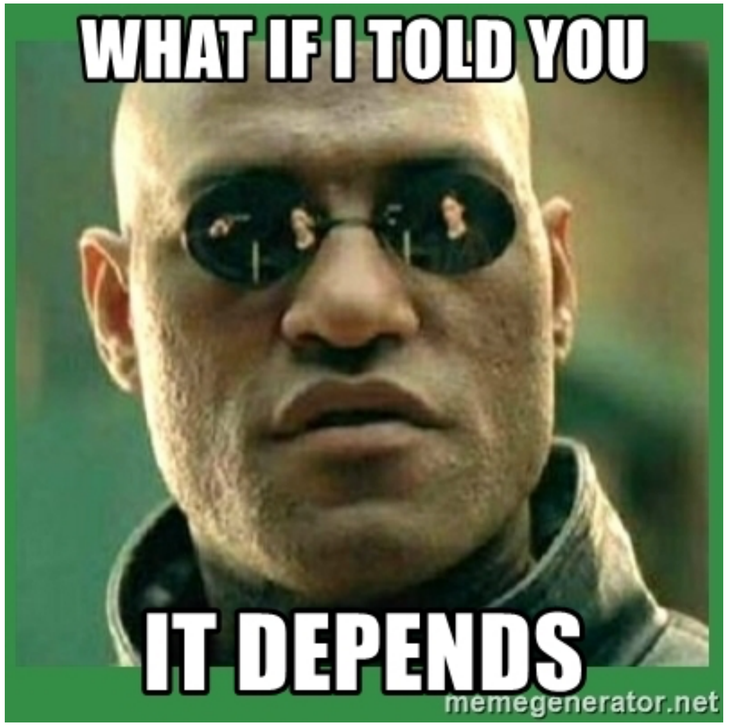
Server-Sent events (aka Event Streams)

Jadwiga Słowik, 13.05.2022

Problem

Jak efektywnie zaimplementować komunikację pomiędzy klientem a serwerem?




WHAT IF I TOLD YOU
IT DEPENDS
memegenerator.net

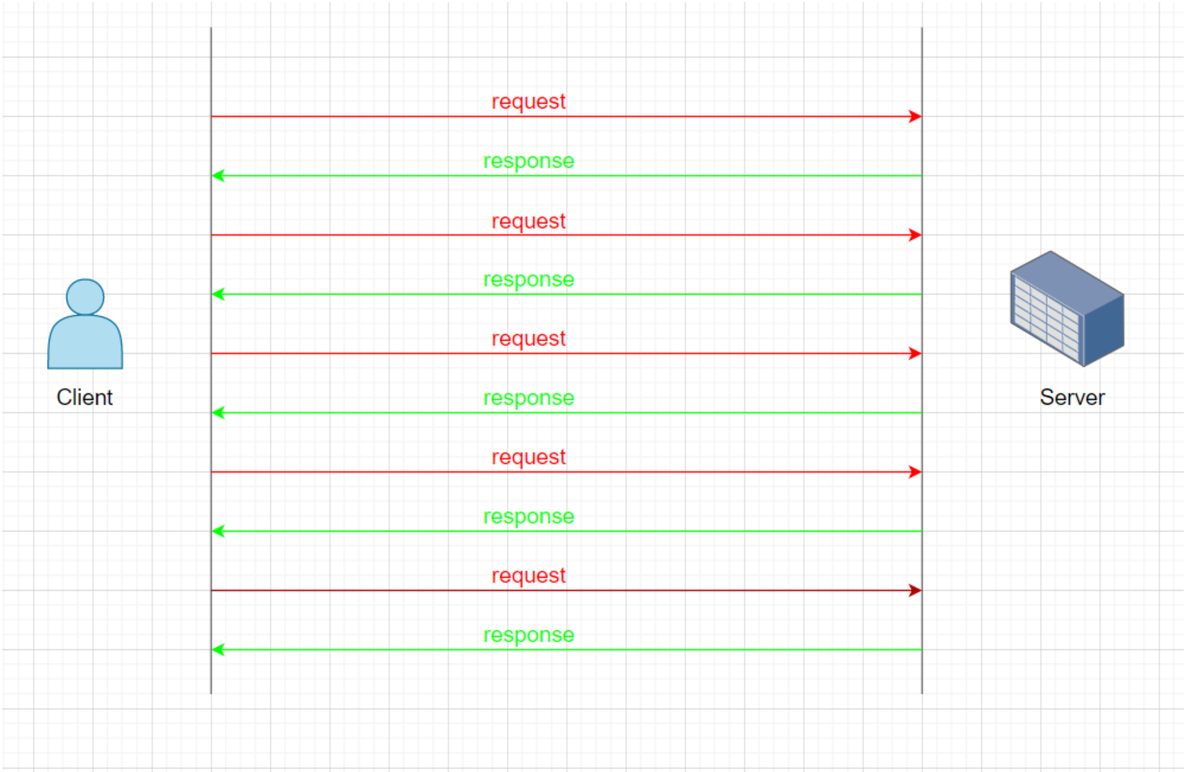
Zależy od konkretnego kontekstu

- Jakie 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?)

Potencjalne rozwiązania

Potencjalne rozwiązania

Http Polling / Short polling



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

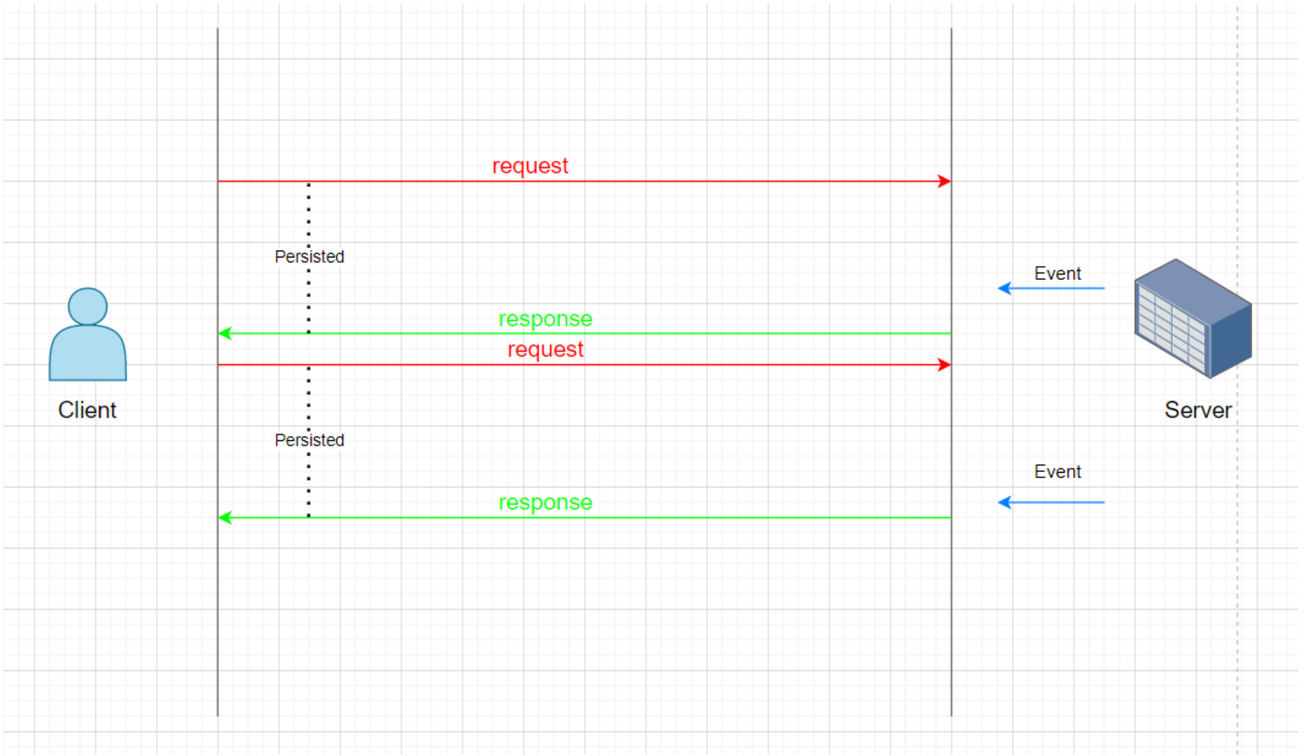
Potencjalne rozwiązania

Http Polling / Short polling

- **Zalety**
  - Prosta implementacja
- **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.

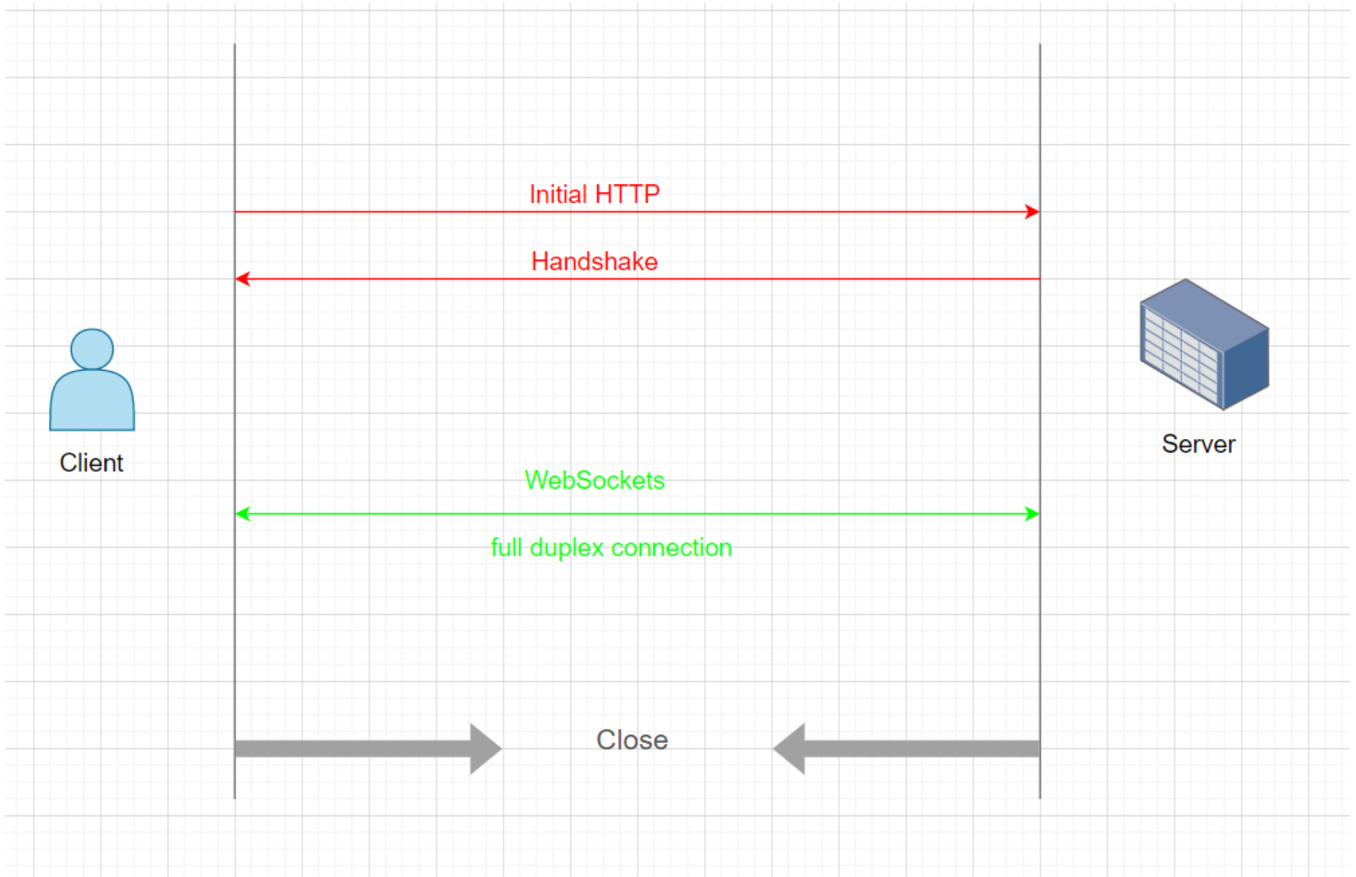
Potencjalne rozwiązania

Http long polling

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

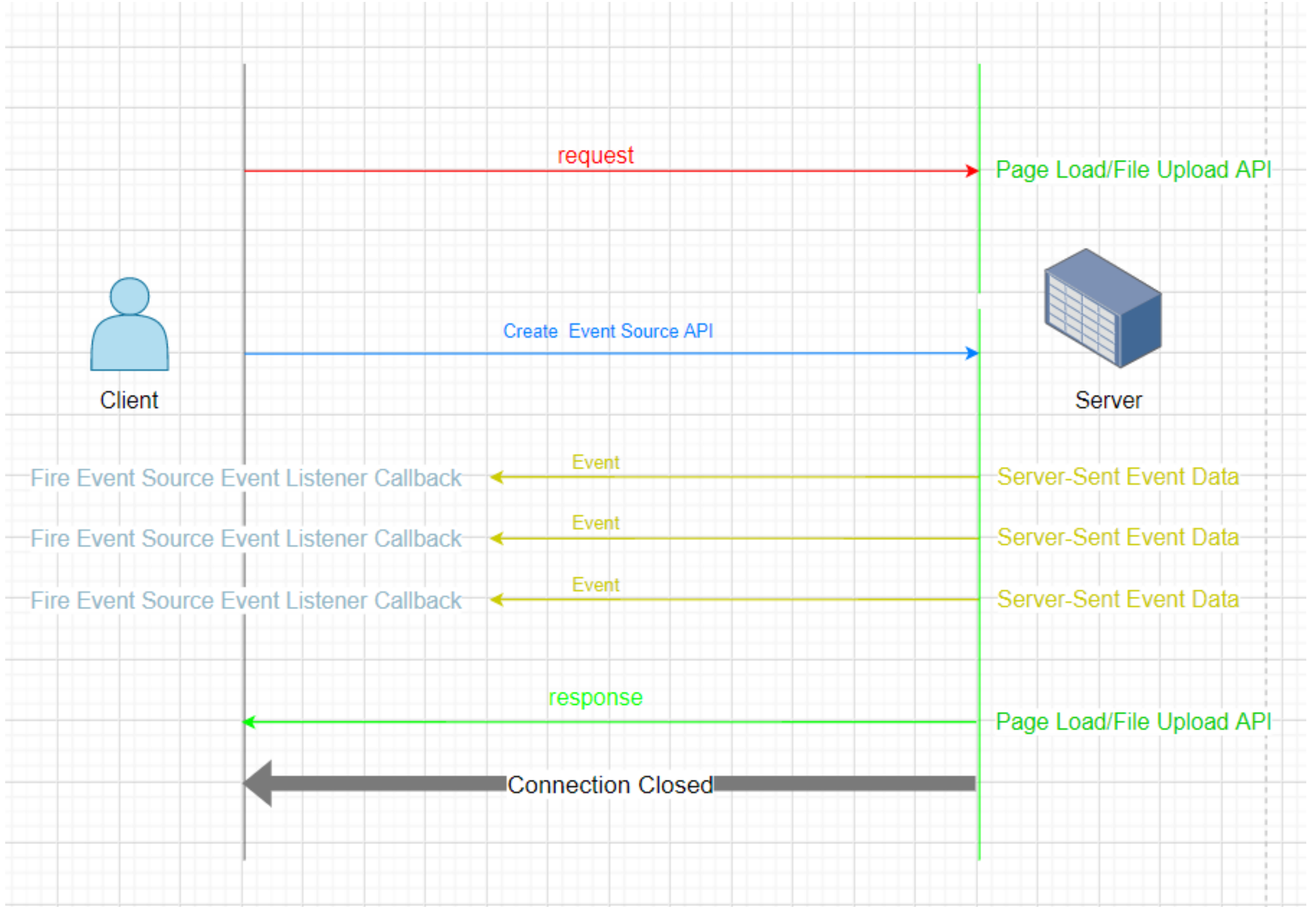
Potencjalne rozwiązania

Web Sockets

- **Zalety**
  - Rzeczywiście implementuje real-time streaming, a nie tylko naśladuje
  - Mniej zapytań, mniej zużycia transferu internetowego
  - Nie ma potrzeby specjalnej obsługi otwartych zapytań przez serwer
- **Wady**
  - Inny protokół, dodatkowe biblioteki

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

- **Zalety**
  - Łatwy w implementacji i prostszy od web sockets
    - Dla Androida: OkHttp-sse
    - Java (Server): SseEmitter
  - Dobra alternatywa dla Web Sockets, gdy nie jest potrzebna obustronna komunikacja
- **Wady**
  - Wiadomości SSE są tekstowe (stąd mniej efektywna obsługa wiadomości binarnych)
  - Jednostronna komunikacja (klient może jedynie coś komunikować w momencie ustanowienia połączenia)
  - Limit na liczbę otwartych połączeń (6 połączeń na przeglądarkę)

Komunikaty SSE

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

SSE - część serwerowa

Ustawienie nagłówków

- ‘Content-Type’: ’text/event-stream’
- ‘Cache-control’: ‘no-cache’
- ‘Connection’: ‘keep-alive’

Web Sockets vs. SSE

Podział zastosowań

Web Sockets	SSE
Wymagające dwustronnej komunikacji	Wystarczająca jest jednostronna komunikacja
Implementacja chatów	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 (<= 6)


TALK IS CHEAP
SHOW ME THE CODE
memegenerator.net