В «джентльменському наборі» будь-якого користувача всесвітньої мережі повинен бути якийсь месенджер (сервіс миттєвого обміну повідомлень). Саме тому їх є незліченна кількість. Це і Telegram, і Whatsapp, і Signal та багато інших. Особливістю програм такого типу є те, що повідомлення від одного користувача до іншого повинно доходити миттєво, тобто в режимі реального часу. І, звісно, користувач має бажання отримати доступ до цього сервісу з будь-якого пристрою, який зараз під рукою у користувача. Саме тому виникла ідея використати для доступу до сервісу в якості клієнта програму, яка завжди існує на всіх пристроях, за допомогою яких користувач доступається до всесвітньої мережі (тобто звичайний браузер).

Проблема та шляхи вирішення

Проблема розробки програми такого типу полягає в тому, що ми створюємо все більш складні веб-додатки на основі протоколу, який, як це не парадоксально, був задуманий і розроблений для набагато простіших форм взаємодії. HTTP не має вбудованої підтримки стану чи навіть безпеки. Його основне припущення полягає в тому, що клієнт надсилає запит, а веб-сервер видає відповідь. Загалом, це означає, що немає запиту, немає відповіді.

**Шляхи вирішення цієї проблеми.**

Існує декілька шляхів вирішення цієї проблеми.

**Періодичні опитування**

Найпростіший спосіб отримати нову інформацію від сервера – періодичний запит. Припустимо, є регулярні запити на сервер вигляду: «Привіт, я тут, у вас є якась інформація для мене?». Наприклад, кожні 10 секунд. У відповідь сервер, по-перше, помічає у себе, що клієнт онлайн, а по-друге посилає весь пакет повідомлень, який накопичений для цього клієнта на цей моменту.

Це працює, але є і недоліки:

Повідомлення передаються із затримкою до 10 секунд (між запитами).

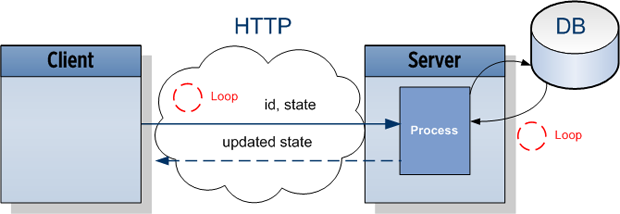
Навіть якщо повідомлень немає, сервер «атакується» запитами кожні 10 секунд, навіть якщо користувач переключився на якусь іншу задачу або спить. С точки зору ефективності, це досить велике навантаження.

Виходячи з цього, якщо мова йде про дуже маленькому сервісі, підхід може виявитися дієздатним, але в цілому він потребує покращення.

**Long pooling (довге опитування)**

Довгі опитування – це найпростіший спосіб підтримувати постійне з'єднання з сервером, не використовуючи ніяких специфічних протоколів (типу WebSocket або Server Sent Events).

Його дуже легко реалізувати, і він добре підходить для багатьох завдань.



Як це працює:

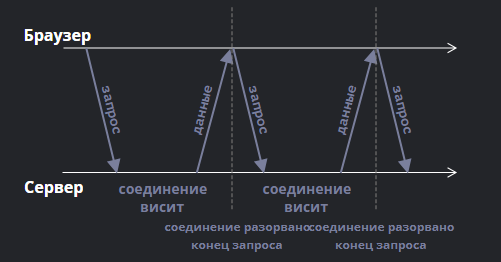
1. Запит надсилається на сервер.

2. Сервер не закриває з'єднання, доки не з'явиться повідомлення для надсилання.

3. Коли з'являється повідомлення – сервер відповідає на запит, надсилаючи його.

4. Браузер негайно надсилає новий запит.

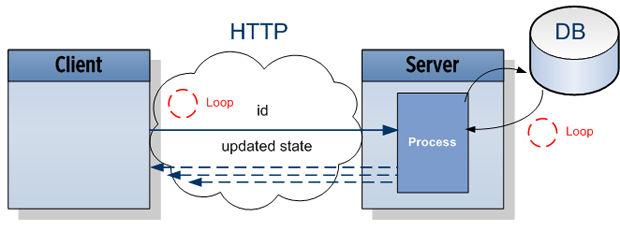
Для цього методу ситуація, коли браузер надіслав запит і утримує з'єднання з сервером в очікуванні відповіді, є стандартною. З'єднання переривається лише доставкою повідомлень.



Якщо з'єднання буде втрачено, скажімо, через мережеву помилку, браузер негайно надсилає новий запит.

Недоліком цього рішення є те, що з'єднання не є постійним і витрачається час на перепідключення.

**Події, надіслані сервером (Server Sent Events)**

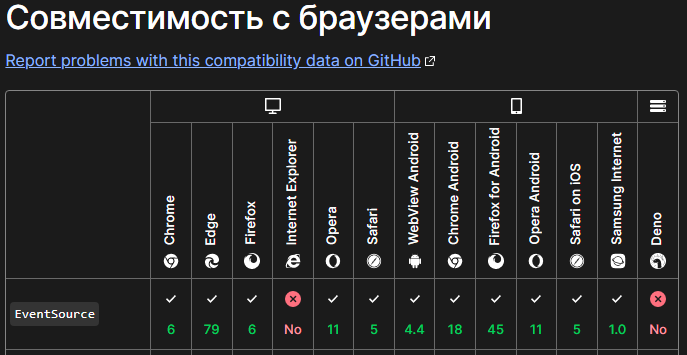


**Як це працює**

Об'єкт EventSource автоматично встановлює постійне з'єднання і дозволяє серверу надсилати повідомлення. Він надає:

* Автоматичне перепідключення з затримкою, що налаштовується.
* Ідентифікатор повідомлень для відновлення з'єднання.
* Останній отриманий ідентифікатор надсилається в заголовку Last-Event-ID під час перепідключенні.
* Поточний стан, записаний у властивості readyState.

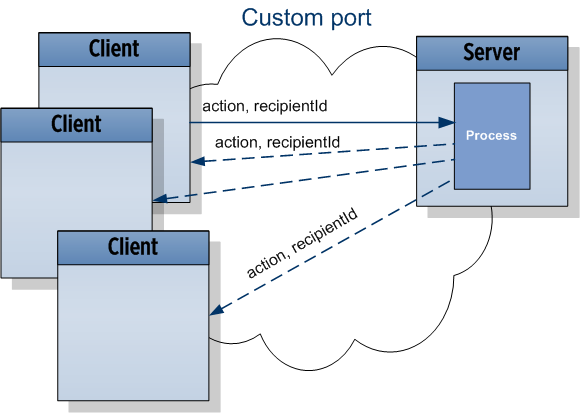
Це робить EventSource гідною альтернативою протоколу WebSocket, який є порівняно низькорівневим і не має таких вбудованих можливостей (хоча їх і можна реалізувати). Для багатьох програм можливостей EventSource цілком достатньо. В той же час існує проблема сумісності із браузерами.



## WebSockets

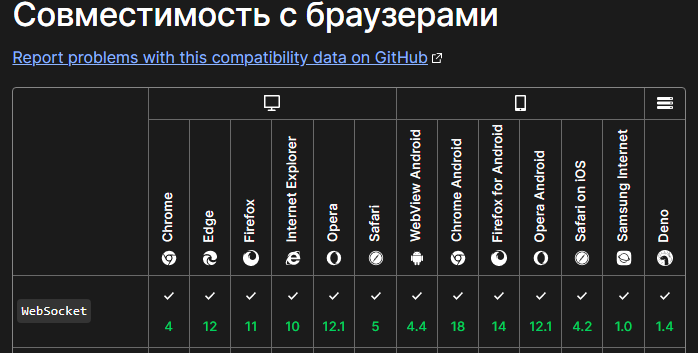
## Протокол WebSocket ("веб-сокет"), описаний у специфікації RFC 6455, забезпечує можливість обміну даними між браузером та сервером через постійне з'єднання. Дані передаються в обох напрямках у вигляді «пакетів», без розриву з'єднання і додаткових HTTP-запитів.

## WebSocket особливо хороший для сервісів, які потребують постійного обміну даними, наприклад онлайн ігри, торгові майданчики, що працюють у реальному часі, і т.д.



### Як це працює

Клієнт сповіщає сервер веб-сокетів (EventMachine) про подію, надаючи ідентифікатори одержувачів. Сервер негайно сповіщає всіх активних клієнтів (підписаних на подію такого типу). Клієнти обробляють подію, коли вказаний ідентифікатор одержувача збігається з ідентифікатором клієнта.



У зв'язку із тим, що на сьогодні дана технологія підтримується всіма сучасними браузерами та є стандартизованою для цього проекту була обрана технологія з використанням обміну повідомленнями за протоколом WebSocket.

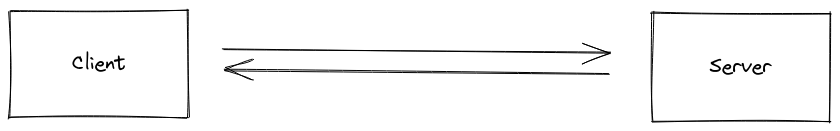
В проекті реалізована як серверна частина програми, так і клієнтська.

В якості рішення для обміну повідомлень використана бібліотека [SocketIO](https://socket.io" \o "https://socket.io).

Це бібліотека, яка забезпечує двосторонній, заснований на подіях зв'язок в режимі реального часу між браузером та сервером. Вона складається із [серверної частини](https://socket.io/docs/v4/server-api/) та [клієнтської бібліотеки Javascript для браузера](https://socket.io/docs/v4/client-api/).

Socket.IO — це бібліотека, яка забезпечує двонаправлений зв’язок між клієнтом і сервером із низькими затримками та на основі подій.

Схема зв'язку між сервером і клієнтом



Він побудований на основі протоколу WebSocket і надає додаткові гарантії, такі як повернення до long-pooling (довгого опитування) HTTP або автоматичне повторне підключення.

Клієнтська частина програми побудована з використанням

