Better WebSockets - Server-Sent Events, a carefree alternative

EuroPython - 12/07/2019 - Andrei Neagu @weetHK
About me (can you guess the city?)

Work as an IT Technical Consultant
I like to travel and explore
Also known as “typo master” at work
Schedule

- SSE introduction
- Inner workings
- Differences from WebSockets
- Implementation explanation for a generic HTTP server in Python
- Some use cases
Raise hands time 🍰👀
Server to client data delivery techniques

- Polling
- LongPolling
- WebSockets
- Server Sent Events
Polling

The dark ages
LongPolling

Slightly less darker
WebSockets

The cool kid, tend to stand out
Server Sent Events

Not that well known

(did you know that a Lavazza museum exist? And that I do not drink coffee?)
Connection wise

Image source: https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9
The simplest example

Javascript

```javascript
const eventStream = new EventSource( url: "/stream");
eventStream.addEventListener( type: "message", listener: message =>{
    // handle message here
});
```

Python

```python
from flask import Flask
app = Flask(__name__)

# Flask
@route("/stream")
def stream():
    def fetch_data():
        for message in BLOCKING_DATA_SOURCE:
            yield 'data: %s

    return Response(fetch_data(), mimetype="text/event-stream")

if __name__ == '__main__':
    app.run()
```
More on EventSource

Available handlers

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Event handler event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>open</td>
</tr>
<tr>
<td>onmessage</td>
<td>message</td>
</tr>
<tr>
<td>onerror</td>
<td>error</td>
</tr>
</tbody>
</table>

Usage in JS

```javascript
const eventStream = new EventSource( url: "/stream" );
eventStream.addEventListener( type: "message", listener: message => {
  // handle message here
}
);
eventStream.addEventListener( type: "error", listener: (error) => {
  // error occurred client side
}
);
eventStream.addEventListener( type: "open", listener: () => {
  // just connected
}
);

// somewhere else, when you are done with it
eventStream.close();
```
More Python frameworks

There are some libraries for django

```python
# TurboGears2
class TheBestController(TGController):
    @expose(content_type='text/event-stream')
    def stream(self, **kwargs):
        def fetch_data():
            for message in BLOCKING_DATA_SOURCE:
                yield 'data: %s

        return fetch_data()

# Pyramid
@view_config(route_name='events')
def stream(request):
    def fetch_data():
        for message in BLOCKING_DATA_SOURCE:
            yield 'data: %s

    headers = [('Content-Type', 'text/event-stream'),
               ('Cache-Control', 'no-cache')]
    response = Response(headerlist=headers)
    response.app_iter = fetch_data()
    return response

# Flask
@route("/stream")
def stream():
    def fetch_data():
        for message in BLOCKING_DATA_SOURCE:
            yield 'data: %s

    return Response(fetch_data(), mimetype='text/event-stream')

# aiohttp
async def stream(request):
    async with sse_response(request) as resp:
        async for message in AWAITABLE_BLOCKING_DATA_SOURCE():
            await resp.send('data: %s

    return resp
```
The internals

* A brief tour
Generic server implementation

Server response headers

Content-Type: text/event-stream
Cache-Control: no-cache
Connection: keep-alive

Body encoding in UTF-8 in the following format

[field]: value\n
Field can have the following values

- data
- event
- id
- retry

: This is a comment ignored by browsers
### Response data format

<table>
<thead>
<tr>
<th>id</th>
<th>data</th>
<th>event</th>
<th>retry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>message1</td>
<td>connected</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>message2</td>
<td>generic unnamed event</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>messageX</td>
<td>disconnected</td>
<td>10000</td>
</tr>
</tbody>
</table>

```
data: 1° message

data: 2° begin message

data: 2° continue message

data: {}

data: "foo": "bar",

data: "baz", 555

data: 

data: User1 just got online

data: generic unnamed event

data: User7 abbandona us

data: 
```
Custom event listeners example/client server

Javascript

```javascript
const eventStream = new EventSource( url: "/stream");
eventStream.addEventListener( type: "greet", listener: message => {
    console.log("Hello ${message.data}");
});
```

Python

```python
from flask import Flask
app = Flask(__name__)

# Flask
@route("/stream")
def stream():
    def fetch_data():
        for name in BLOCKING_DATA_SOURCE:
            yield 'event: greet\ndata: %s\n\n' % name

        return Response(fetch_data(), mimetype="text/event-stream")

if __name__ == '__main__':
    app.run()
```
More on SSE

- Requests can be redirected HTTP 301 (permanent) & 307 (temporary)
- Only UTF-8 decoding is supported, no binary data
- Protocol supports multiple type of events, default is message
- Clients always reconnect (no need to handle)
- Server sends HTTP 204 No Content to stop reconnection
- Limited amount of global connections per site
Native browser support

Source: https://caniuse.com/#feat=eventsource 09/07/2019
Other browsers, via polyfill https://github.com/Yaffle/EventSource
Can I use it without a browser?
Yes, there are libraries

- Python: sseclient, sseclient-py, aiosseclient
- Android: sse-android, RxSSE
- iOS: EventSource (Swift), ios-eventsource (Objective-C)
- react-native: react-native-event-source (based on a polyfill)
SSE vs WebSockets

- Only UTF-8 encoding
- Uses HTTP
- Proxy friendly
- Built-in support for reconnection and synchronization
- Detects disconnection server side when trying to send out data
- Only Server -> Client data channel
- Clients automatically handle disconnections by reconnecting
- Also supports binary data
- Has a custom protocol
- May have to reconfigure some proxies
- Heartbeat, does not always work
- Can detect disconnections server side
- Can send data in both directions
- Client disconnections must be explicitly handled
Use cases

- Dashboards
- News feeds
- Notifications to browser
- Games (depends on the game)
Some possibly useful links

- https://www.w3.org/TR/eventsource/
- http://html5doctor.com/server-sent-events/
- https://streamdata.io/blog/push-sse-vs-websockets/
Takeaways

- Consider SSE for your next project
- Choose between SSE and WebSockets as it makes sense for your application
Thank you!

Questions?

Contact me on Twitter @weetHK

All the pictures used in this presentation are places from or near Turin