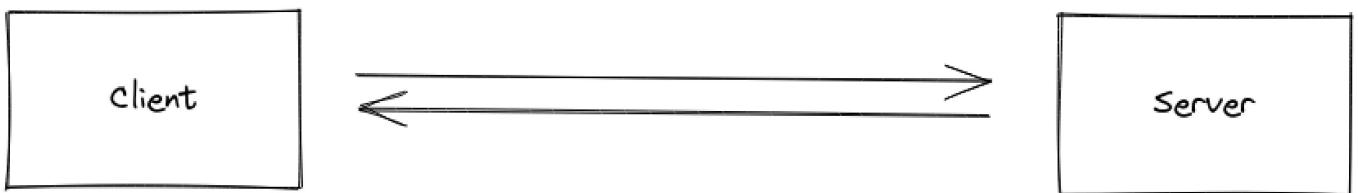


**Version: 4.x**[On this page](#)

# Introduction

## What Socket.IO is

Socket.IO is a library that enables **low-latency**, **bidirectional** and **event-based** communication between a client and a server.



It is built on top of the [WebSocket](#) protocol and provides additional guarantees like fallback to HTTP long-polling or automatic reconnection.

**ⓘ INFO**

WebSocket is a communication protocol which provides a full-duplex and low-latency channel between the server and the browser. More information can be found [here](#).

There are several Socket.IO server implementations available:

- JavaScript (Node.js) (whose documentation can be found here on this website)
  - [Installation steps](#)
  - [API](#)
  - [Source code](#)
- JavaScript (Deno): <https://github.com/socketio/socket.io-deno>
- Java: <https://github.com/mrniko/netty-socketio>
- Java: <https://github.com/trinopaty/socket.io-server-java>
- Python: <https://github.com/miguelgrinberg/python-socketio>
- Golang: <https://github.com/googollee/go-socket.io>

And client implementations in most major languages:

- JavaScript (which can be run either in the browser, in Node.js or in React Native)

- Installation steps
  - API
  - Source code
- JavaScript (for WeChat Mini-Programs): <https://github.com/weapp-socketio/weapp.socket.io>
  - Java: <https://github.com/socketio/socket.io-client-java>
  - C++: <https://github.com/socketio/socket.io-client-cpp>
  - Swift: <https://github.com/socketio/socket.io-client-swift>
  - Dart: <https://github.com/rikulo/socket.io-client-dart>
  - Python: <https://github.com/miguelgrinberg/python-socketio>
  - .Net: <https://github.com/doghappy/socket.io-client-csharp>
  - Rust: <https://github.com/1c3t3a/rust-socketio>
  - Kotlin: <https://github.com/icerockdev/moko-socket-io>

Here's a basic example with plain WebSockets:

*Server* (based on `ws`)

```
import { WebSocketServer } from "ws";

const server = new WebSocketServer({ port: 3000 });

server.on("connection", (socket) => {
  // send a message to the client
  socket.send(JSON.stringify({
    type: "hello from server",
    content: [ 1, "2" ]
  }));

  // receive a message from the client
  socket.on("message", (data) => {
    const packet = JSON.parse(data);

    switch (packet.type) {
      case "hello from client":
        // ...
        break;
    }
  });
});
```

*Client*

```

const socket = new WebSocket("ws://localhost:3000");

socket.addEventListener("open", () => {
  // send a message to the server
  socket.send(JSON.stringify({
    type: "hello from client",
    content: [ 3, "4" ]
  }));
});

// receive a message from the server
socket.addEventListener("message", ({ data }) => {
  const packet = JSON.parse(data);

  switch (packet.type) {
    case "hello from server":
      // ...
      break;
  }
});

```

And here's the same example with Socket.IO:

### *Server*

```

import { Server } from "socket.io";

const io = new Server(3000);

io.on("connection", (socket) => {
  // send a message to the client
  socket.emit("hello from server", 1, "2", { 3: Buffer.from([4]) });

  // receive a message from the client
  socket.on("hello from client", (...args) => {
    // ...
  });
});

```

### *Client*

```
import { io } from "socket.io-client";
```

```
const socket = io("ws://localhost:3000");

// send a message to the server
socket.emit("hello from client", 5, "6", { 7: Uint8Array.from([8]) });

// receive a message from the server
socket.on("hello from server", (...args) => {
  // ...
});
```

Both examples looks really similar, but under the hood Socket.IO provides additional features that hide the complexity of running an application based on WebSockets in production. Those features are listed [below](#).

But first, let's make it clear what Socket.IO is not.

## What Socket.IO is not

### CAUTION

Socket.IO is **NOT** a WebSocket implementation.

Although Socket.IO indeed uses WebSocket for transport when possible, it adds additional metadata to each packet. That is why a WebSocket client will not be able to successfully connect to a Socket.IO server, and a Socket.IO client will not be able to connect to a plain WebSocket server either.

```
// WARNING: the client will NOT be able to connect!
const socket = io("ws://echo.websocket.org");
```

If you are looking for a plain WebSocket server, please take a look at [ws](#) or [μWebSockets.js](#).

There are also [discussions](#) for including a WebSocket server in the Node.js core.

On the client-side, you might be interested in the [robust-websocket](#) package.

### CAUTION

Socket.IO is not meant to be used in a background service for mobile applications.

The Socket.IO library keeps an open TCP connection to the server, which may result in a high battery drain for your users. Please use a dedicated messaging platform like [FCM](#) for this use case.

## Features

Here are the features provided by Socket.IO over plain WebSockets:

### HTTP long-polling fallback

The connection will fall back to HTTP long-polling in case the WebSocket connection cannot be established.

This feature was the #1 reason people used Socket.IO when the project was created more than ten years ago (!), as the browser support for WebSockets was still in its infancy.

Even if most browsers now support WebSockets (more than [97%](#)), it is still a great feature as we still receive reports from users that cannot establish a WebSocket connection because they are behind some misconfigured proxy.

### Automatic reconnection

Under some particular conditions, the WebSocket connection between the server and the client can be interrupted with both sides being unaware of the broken state of the link.

That's why Socket.IO includes a heartbeat mechanism, which periodically checks the status of the connection.

And when the client eventually gets disconnected, it automatically reconnects with an exponential back-off delay, in order not to overwhelm the server.

### Packet buffering

The packets are automatically buffered when the client is disconnected, and will be sent upon reconnection.

More information [here](#).

### Acknowledgements

Socket.IO provides a convenient way to send an event and receive a response:

### Sender

```
socket.emit("hello", "world", (response) => {
  console.log(response); // "got it"
});
```

### Receiver

```
socket.on("hello", (arg, callback) => {
  console.log(arg); // "world"
  callback("got it");
});
```

You can also add a timeout:

```
socket.timeout(5000).emit("hello", "world", (err, response) => {
  if (err) {
    // the other side did not acknowledge the event in the given delay
  } else {
    console.log(response); // "got it"
  }
});
```

## Broadcasting

On the server-side, you can send an event to **all connected clients** or **to a subset of clients**:

```
// to all connected clients
io.emit("hello");

// to all connected clients in the "news" room
io.to("news").emit("hello");
```

This also works when **scaling to multiple nodes**.

## Multiplexing

Namespaces allow you to split the logic of your application over a single shared connection. This can be useful for example if you want to create an "admin" channel that only authorized users can join.

```
io.on("connection", (socket) => {
  // classic users
});

io.of("/admin").on("connection", (socket) => {
  // admin users
});
```

More on that [here](#).

## Common questions

### Is Socket.IO still needed today?

That's a fair question, since WebSockets are supported [almost everywhere](#) now.

That being said, we believe that, if you use plain WebSockets for your application, you will eventually need to implement most of the features that are already included (and battle-tested) in Socket.IO, like [reconnection](#), [acknowledgements](#) or [broadcasting](#).

### What is the overhead of the Socket.IO protocol?

`socket.emit("hello", "world")` will be sent as a single WebSocket frame containing  
`42["hello","world"]` with:

- `4` being Engine.IO "message" packet type
- `2` being Socket.IO "message" packet type
- `["hello","world"]` being the `JSON.stringify()`-ed version of the arguments array

So, a few additional bytes for each message, which can be further reduced by the usage of a [custom parser](#).



The size of the browser bundle itself is [10.4 kB](#) (minified and gzipped).

## Something does not work properly, please help?

Please check the [Troubleshooting guide](#).

## Next steps

- [Get started example](#)
- [Server installation](#)
- [Client installation](#)

 [Edit this page](#)

*Last updated on 12/19/2022*