Task 4

# Stream Data from a Networked Service

[**Interfaces**](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API#interfaces)

[WebSocket](https://developer.mozilla.org/en-US/docs/Web/API/WebSocket)

The primary interface for connecting to a WebSocket server and then sending and receiving data on the connection.

[CloseEvent](https://developer.mozilla.org/en-US/docs/Web/API/CloseEvent)

The event sent by the WebSocket object when the connection closes.

[MessageEvent](https://developer.mozilla.org/en-US/docs/Web/API/MessageEvent)

The event sent by the WebSocket object when a message is received from the server.

[**Guides**](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API#guides)

* [Writing WebSocket client applications](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications)
* [Writing WebSocket servers](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers)
* [Writing a WebSocket server in C#](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_server)
* [Writing a WebSocket server in Java](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_a_WebSocket_server_in_Java)

[**Tools**](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API#tools)

* [AsyncAPI](https://www.asyncapi.com/): A specification for describing event-driven architectures like WebSocket. You can use it to describe WebSocket-based APIs just as you would describe REST APIs with the OpenAPI specification. Learn [why you should consider using AsyncAPI with WebSocket](https://www.asyncapi.com/blog/websocket-part1) and [how to do so](https://www.asyncapi.com/blog/websocket-part2).
* [HumbleNet](https://hacks.mozilla.org/2017/06/introducing-humblenet-a-cross-platform-networking-library-that-works-in-the-browser/): A cross-platform networking library that works in the browser. It consists of a C wrapper around WebSockets and WebRTC that abstracts away cross-browser differences, facilitating the creation of multi-user networking functionality for games and other apps.
* [µWebSockets](https://github.com/uNetworking/uWebSockets): Highly scalable WebSocket server and client implementation for [C++11](https://isocpp.org/) and [Node.js](https://nodejs.org/).
* [Socket.IO](https://socket.io/): A long polling/WebSocket based third party transfer protocol for [Node.js](https://nodejs.org/).
* [SocketCluster](https://socketcluster.io/): A pub/sub WebSocket framework for [Node.js](https://nodejs.org/) with a focus on scalability.
* [WebSocket-Node](https://github.com/theturtle32/WebSocket-Node): A WebSocket server API implementation for [Node.js](https://nodejs.org/).
* [Total.js](https://www.totaljs.com/): Web application framework for [Node.js](https://nodejs.org/en/) (Example: [WebSocket chat](https://github.com/totaljs/examples/tree/master/websocket))
* [Faye](https://www.npmjs.com/package/faye-websocket): A [WebSocket](https://developer.mozilla.org/en-US/docs/Web/API/WebSocket) (two-ways connections) and [EventSource](https://developer.mozilla.org/en-US/docs/Web/API/EventSource) (one-way connections) for [Node.js](https://nodejs.org/) Server and Client.
* [SignalR](https://dotnet.microsoft.com/en-us/apps/aspnet/signalr): SignalR will use WebSockets under the covers when it's available, and gracefully fallback to other techniques and technologies when it isn't, while your application code stays the same.
* [Caddy](https://caddyserver.com/): A web server capable of proxying arbitrary commands (stdin/stdout) as a websocket.
* [ws](https://github.com/websockets/ws): a popular WebSocket client & server library for [Node.js](https://nodejs.org/).
* [jsonrpc-bidirectional](https://github.com/bigstepinc/jsonrpc-bidirectional): Asynchronous RPC which, on a single connection, may have functions exported on the server and, and the same time, on the client (client may call server, server may also call client).
* [cowboy](https://github.com/ninenines/cowboy): Cowboy is a small, fast and modern HTTP server for Erlang/OTP with WebSocket support.
* [WebSocket King](https://websocketking.com/): A client tool to help develop, test and work with WebSocket servers.
* [PHP WebSocket Server](https://github.com/napengam/phpWebSocketServer): Server written in PHP to handle connections via websockets wss:// or ws://and normal sockets over ssl:// ,tcp://
* [Channels](https://channels.readthedocs.io/en/stable/index.html): Django library that adds support for WebSockets (and other protocols that require long running asynchronous connections).
* [Flask-SocketIO](https://flask-socketio.readthedocs.io/en/latest/): gives Flask applications access to low latency bi-directional communications between the clients and the server.
* [Gorilla WebSocket](https://pkg.go.dev/github.com/gorilla/websocket): Gorilla WebSocket is a [Go](https://go.dev/) implementation of the WebSocket protocol.

[**Related Topics**](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API#related_topics)

* [AJAX](https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX)
* [JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript)