Here’s a scenario: I’m playing an online multiplayer shooter video game and I see an enemy player coming within shooting range. I slowly set up my shot~~;~~: point, aim, and… shoot. I ~~for sure~~ was certain I got the kill~~,~~ - no question about it. However, I disappointedly discover ~~that~~ my opponent has moved after I pulled the trigger. How did this ~~have~~ happen?

This is where WebSockets come into play.

So, what is this magic~~al~~ ~~thing we~~ called a WebSocket?

WebSockets make it possible to create an interactive communication session between a user’s browser and a server. ~~What that~~ It allows ~~is~~ a bi-directional flow of data so a user’s browser only has to listen for data instead of requesting it every time. Amazing…but what does that mean? To use our example of a multiplayer game, I shouldn’t have to request data on player deaths, current team score, and player position every few seconds to get the updated data (otherwise that would make it very difficult to predict where I should aim). I should just be able to listen to the server and it will send me event-driven responses when it’s available (real-time), not when I request it.

Now that we have an idea of how a WebSocket can work in a real-world situation, let’s dive a bit deeper into the inner workings.

Referring to the diagram above, and also the “GET” request below, the client first makes an initial request to the server through an HTTP(HyperText Transfer Protocol) request. HTTP is a client-server protocol (requests initiated by the browser) which allows the fetching of resources. Resources being content such as HTML files, javascript, and images.

This request is coupled with an “Upgrade” header, which signals to the server that it wants to make a WebSocket connection. Together, this is called the “handshake”. Once the handshake is accepted, as in the server supports the protocol, the initial HTTP request is replaced with a WebSocket connection — a single TCP(Transmission Control Protocol) that defines how to establish and maintain a network connection. With the WebSocket connection in place, this allows the server to push data to the client as the data becomes available.

WebSockets provide a ton of flexibility for developers in terms of freedom of creativity with their apps. Since the client only has to listen for events because of the open connection, lower latency rates and the concept of providing real-time data to clients makes applications relevant to everyday needs.

A few everyday applications that rely on real-time data:

* TSX/NYSE-financial tickers
* Chat rooms
* Online multiplayer games
* Social feeds
* Sport stats updates
* Location-based apps

Implementation

Now it’s time to build a real-time app. Depending on the size of the app and the language it’s being built-in, there’s a WebSocket for everyone. Let’s explore a few tools for Node.js.

socket.io

Want to build a chat feature into your app? Socket.io is the library for you. Easy implementation with no prior knowledge of Node.js required.

most popular WebSocket library

provides the basic framework for easy set-up and integration

lots of documentation and descriptions to make it easy for a beginner to follow

works on every platform, browser, and device

supports real-time analytics, binary streaming, real-time document collaboration, and instant messaging/chat.

slack group provided if any user has any questions or needs help with socket.io

ws

A “simple to use, blazing-fast, and thoroughly tested WebSocket client and server implementation” for Node.js.

Requires a bit more knowledge to implement (not as much detail provided for implementation)

used by over 2,450,000 people!

One of the fastest libraries, if not the fastest.

primus

Don’t know what to use? Why not pick both! Primus is “universal wrapper for real-time frameworks”. Primus works by switching between real-time frameworks and debugging when one or more fail.

It‘s’ simple low-level interface is compatible with the Node.js streams so you can transfer data back and forth.

provides effortless switching between real-time frameworks by changing only one line of code.

Built-in reconnect — reduces server stress.

Offline detection: detects when a user drops their connection~~,~~ and reconnects when they are back online.

documentation for installation and terms is extensive, thorough and clear!

Primus does the heavy lifting and allows the developer to focus on building your application/module without having to worry about the framework that’s best suited for their project.

Which one do I choose? Depends on the scale of the app being created and what documentation the developer finds most helpful.

What about paid + managed solutions for WebSockets?

Pusher

A hosted API that helps developers build scalable real-time features. Build a chat app in a matter of hours, not days.

Comes with a UI kit that provides ready to implement messaging components.

Provides a multitude of use-cases for different types of apps.

Pusher “Channels” provide libraries for web browsers, iOS and Android apps, PHP frameworks, cloud functions, bash scripts, and IoT devices

All in all, there is a WebSocket library for everyone, for all levels of developers. The main takeaway is that WebSockets provide flexibility, creativity, and relevant data for the user and developer, while also establishing and maintaining fast and reliable connections between the server and client.