Technology report: WebSockets

Freddie Askem

March 5th

Rationale

When the World Wide Web first came into existence, all sites were completely static. That is to say, they had no content that was updated after page load. After some time, developers wished to be able to update elements dynamically, and one result of this was JavaScript, a scripting language in the browser. Of most relevance here its the ability to perform requests to the server, and then use the response to update page content. This enabled a far more interactive and smooth web experience as users no longer needed to refresh to receive updates to the page. However, the capabilities of the request-response design of HTTP became limiting, as there is no way for the server to alert the client of changes that it may be interested in. Instead, the client can only receive new content when it specifically requests it. This led to the rise of polling, where clients would regularly send requests to the server (say every 5 seconds) to see if there are any updates. This process wastes bandwidth, client resources, and server resources. It would therefore be much more convenient for the server to be able to push new content to the client as it becomes available. While workarounds such as HTTP long polling were used, these were not proper solutions to the issue and had their own problems. The real solution to this issue came in the form of WebSockets, which allow fully asynchronous duplex communication.

WebSockets

As alluded to in the previous section, WebSockets are asynchronous. This means that events can arrive and be transmitted at any point in time without any guarantee of ordering. While this makes them very powerful, it can make programming with them challenging. To manage this complexity, most systems employ an event-driven architecture. What this means is that arriving WebSocket messages trigger events (functions), and the program is designed in a way that it is safe for those particular functions to execute at any time. Likewise, when sending a message most systems do not wait for a response, and instead immediately continue with other tasks, allowing any arriving messages to be processed by the event handlers. WebSockets also allow arbitrary data to be transferred, though many systems pick a standardised format such as JSON to permit easier interoperation.

Application

In our project, there are a few components that would benefit from live updates. Chiefly, we should implement notifications for incoming connection requests, live connection list updates as others accept requests or remove connections, and render connection graph updates live. WebSockets are the ideal technology for this as changes to users' connections can happen at any time. Since we require WebSockets for core functionality, this was the primary driver behind our decision to switch technology stack. This is as the JHipster/Spring stack does not support WebSockets well.

Implementation

The implementation divides into three parts: the reverse proxy (nginx), the server (nodejs), and the client (web browsers). The reverse proxy requires a new route to be added that supports upgrading the connection to a WebSocket. This is achieved using the `Upgrade` HTTP header. The server should use the npm (node package manager) library `ws`, which is the de facto WebSocket library for nodejs. This library provides an API in the style of `socket.on(event, function)`. In other words, this says "upon some event on the WebSocket, execute this function". Since we use React on the client, it is sensible to use a WebSocket library that is specifically designed to work with React's model of the DOM. Therefore, I recommend using `react-use-websocket` for this purpose.

Authentication and security

Finally, we must be aware of the security implications of using WebSockets. We need to ensure that connection change events are only sent to the involved users while respecting privacy settings, as this would otherwise reveal information about people's connections to users who should not have knowledge of them. We must also ensure that connections to the WebSocket API we expose fully authenticate the user connecting, as otherwise it is possible to impersonate other users and access their information.

Resources

- https://github.com/robtaussig/react-use-websocket
- https://blog.logrocket.com/websocket-tutorial-real-time-node-react/
- https://www.npmjs.com/package/ws