Front end Javascript with Socket.io

(Version 2016-12-02)

Socket.io is a bidirectional, event based, real time communication method that operates on the web socket layer. Web apps created with Socket.io benefit from streamlined development but also with unique challenges.

Advantages:

1) Very easy to setup.

- A server with Socket.io enabled will automatically stream a client version of the module to the webpage requested.

2) Bi-directional, event based.

- Sockets can listen and send events at any time. Once a listener of an event is installed in the client-side, that event emitted from the server will always be caught by that listener.

3) Global Socket Object.

- Once a socket variable is assigned to the object returned by a successful socket connection, that socket variable can be used throughout the entire code.

Disadvantages:

1) Page based connection.

- A socket connection is only valid throughout the current html page. If the URL changes, without additional measures to re-establish the same session, that connection is lost.

How to initialize:

Stick the script tag in the <head> section of an HTML document to get the base for Socket.io.

```
<script src="/socket.io/socket.io.js"></script>
```

In a separate script tag or javascript file,

```
var socket = io.connect( );
```

to establish a connection between the server and the client web socket.

A good practice is to declare a global variable where the socket will be assigned to, but do the actual connection once the document or window is done loading to prevent any race conditions.

```
var socket;
window.onload = function () { socket = io.connect(); }
```

How to listen to server events:

The basic syntax for adding a socket event listener is as follows.

```
socket.on('event_name', function (arguments) {
    // Callback function code...
});
```

Event Name: String that represents the event name.
Callback : Function that runs with the event's data passed in.

A callback function is required for any socket event listener, but the existence of passed in arguments is optional. A socket event that notifies when a registration went successful doesn't need to provide the client with any data other than the actual notification itself. On the other hand, when the result of a penetration test needs to be sent back to the client, the data is passed as one of the arguments of the callback function.

The callback function can have multiple arguments specific to the event, so specifications for each event should be provided by the server API.

How to emit an event to the server:

The basic syntax for emitting a socket event is as follows.

```
socket.emit('event_name', data1, data2, ...);
```
Again, whether you need to pass in any data with the event is optional. Data can be of any form, string, JSON, array, number, etc.

Uniqueness of each socket connection:

In the client-side, connecting to the server is straightforward. However, Each socket connection provides a unique connection session that is internal to the Socket.io module, and this unique session ability is used extensively in the server-side.
A socket will be assigned a unique id (ex. 3v7DZ1iR-kEt5biMAAAE) upon connection, and the server will assign properties specific to that socket object. For example, upon user login, the server will create a small session object and assign it to socket.session. Of course, the client-side won't be able to view this data since its specific to the server's implementation, but once this connection is lost, even when trying to reconnect from the same browser after a page change, a new socket connection with a new socket id will be established, severing any connections it had with the previous session that contained user credentials.

How to circumvent the disconnect problem:

As mentioned earlier, if a user moves to an entirely different page within the website, then the current socket connection disappears, losing track of session data stored in the server. In order to keep a user intact throughout the entirety of the website, there are some possible solutions.

The first solution is to use the server's cookie session along with Socket.io. Since cookies can be programmed to persist throughout a predetermined time, we can use the session cookie to prove user authentication and quickly restore the socket session onto the new socket. However, one downside to this is that it is relatively inefficient to do so because each page change will need to repeat the authentication process and it can be quite a load to the server.

Another solution is to simply keep all the content inside one HTML page and use frameworks such as Jquery to have "soft" page changes. This means that all of the HTML content of the website will be in one file, but render only one page container at a time. With non-AJAX soft page changes, a page change such as
```
window.location.assign("#otherpage")
```
will look like it is going to a different page, but it is just transitioning to another page container called 'otherpage' within the same file. This adds the obvious benefit that one socket connection will persist throughout the whole website, but may cause some flexibility issues in web development.