WEBSOCKETS

Josiah Mory / @kickinbahk

Standard REST Request dissected

Request a Website

http://www.meetup.com/Riverside-Ruby-User-Group/

Standard Rest Transaction

1. Open a socket port to 80 on meetup.com
2. Send an Http header request to the server (Apache/Nginx)
3. This buffers the message
4. Message is sent to server apppcation
5. Server application decides what to do with request

Standard Rest Transaction (cont)

6. Fetches Data
7. Generates HTML
8. Sends it back to the server (Apache/Nginx)
9. Appropriate HTTP headers are added to the body
10. Sent back to browser and connection closed

Cookies

Identity Maintained by Cookies

Passed back and forth with all requests

Disadvantages

- Carries Overhead
- Open to Security Vulnerabilities

Introducing Websockets

Originally Part of HTML5 standard

Moved to its own standard to keep the specification focused

Bi-directional, full-duplex persistent connection from a web browser to a server

An Interactive communication session between the user's browser and a server

Client or Server can pass messages to each other at any time

Remains Open till Client Closes Connection

Stateless

No Connection Limitation

(Mulitple Tabs)

Faye - Rails
Socket.io - Node
ActionCable - Rails

Faye - Rails

Socket.io - Node

ActionCable - Rails

Calling the Websockets API

```
var connection = new WebSocket(url, [protocol] );
```

Ex:

```
var connection = new WebSocket(
                     'ws://html5rocks.websocket.org/echo',
                     ['soap', 'xmpp']);
```

Second Argument

-

Accepts Optional Subprotocols as a String or an Array of Strings

Events

```
// When the connection is open,
// send some data to the server
connection.onopen = function () {
  // Send the message 'Ping' to the Server
  connection.send('Ping');
};

// Log errors
connection.onerror = function (error) {
  console.log('WebSocket Error ' + error);
};
```

Events (cont.)

```
// Receive data from the server
connection.onmessage = function (e) {
  console.log('Server: ' + e.data);
};

// Close connection
connection.onclose = function (c) {
  console.log('Connection: closed...' + c)
};
```

Sending Data

```
// Sending a string
connection.send('your message');

// Sending canvas ImageData as ArrayBuffer
var img = canvas_context.getImageData(0, 0, 400, 320);
var binary = new Uint8Array(img.data.length);
for (var i = 0; i < img.data.length; i++) {
  binary[i] = img.data[i];
}
connection.send(binary.buffer);

// Sending file as Blob
var file = document.querySelector('input[type="file"]').files
connection.send(file);
```

Possible Use Cases

- Multiplayer online games
- Chat applications
- Live sports ticker
- Realtime updating social streams

Why Not Websockets?

Users want *"delightful realtime web apps".*

Developers want *"delightfully easy to build realtime web apps".*

Operations want *"delightfully easy to deploy, scale and manage realtime web apps".*

Difficult to Implement

Difficult to Debug

No Connection Limitation

Complicated Load Balancing

Illusion of Reliability

Lack of Browser Support

Other ways for Similar Results

Long Polling

- Facebook
- Gmail

HTTP/2 Protocol with Long Polling

- Twitter

https://samsaffron.com/archive/2015/12/29/websockets-caution-required

Action Cable/Faye/Socket.io
is Abstraction on WebSockets API

Allows for built-in Functionality

But Functionality Possible in other Ways, with More Benefits

Know What Problem it will Solve for Me

Questions?