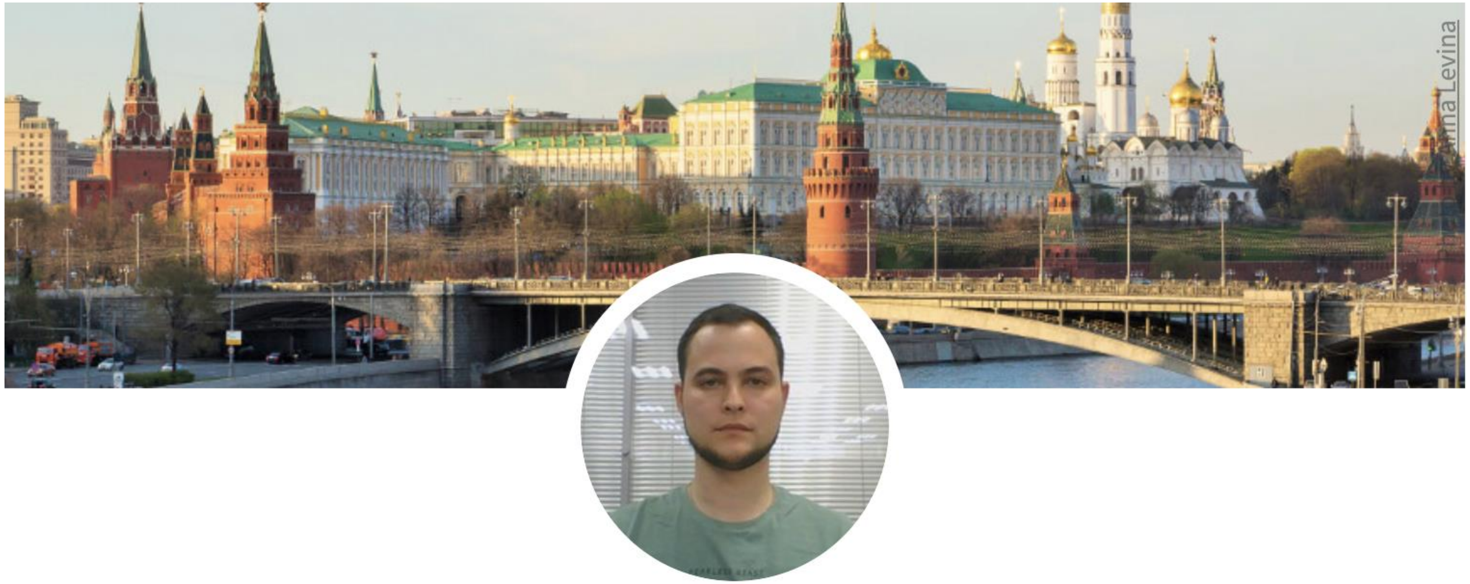
Handling HTTP and WebSocket Protocol

Overview

- Hypertext Transfer Protocol
- Node.js HTTP module
- Node.js and WebSocket Brief

Self-Introduction



Sergei Orlov

Senior Software Engineer

Russia, Moscow, 1-y Nagatinskiy proezd, 10, str.1, Floor 3-1, 3.1.34

On vacation from 15 Dec till 31 Dec

Hypertext Transfer Protocol

HTTP STRUCTURE

- HEADER

The tough part: header includes headers

NOTE: HTTP Request requires additional ***HOST*** header which makes it not that additional

NOTE 2: Commonly, HTTP parsers refer to URN, HTTP Method & HTTP Version as headers although they technically aren't

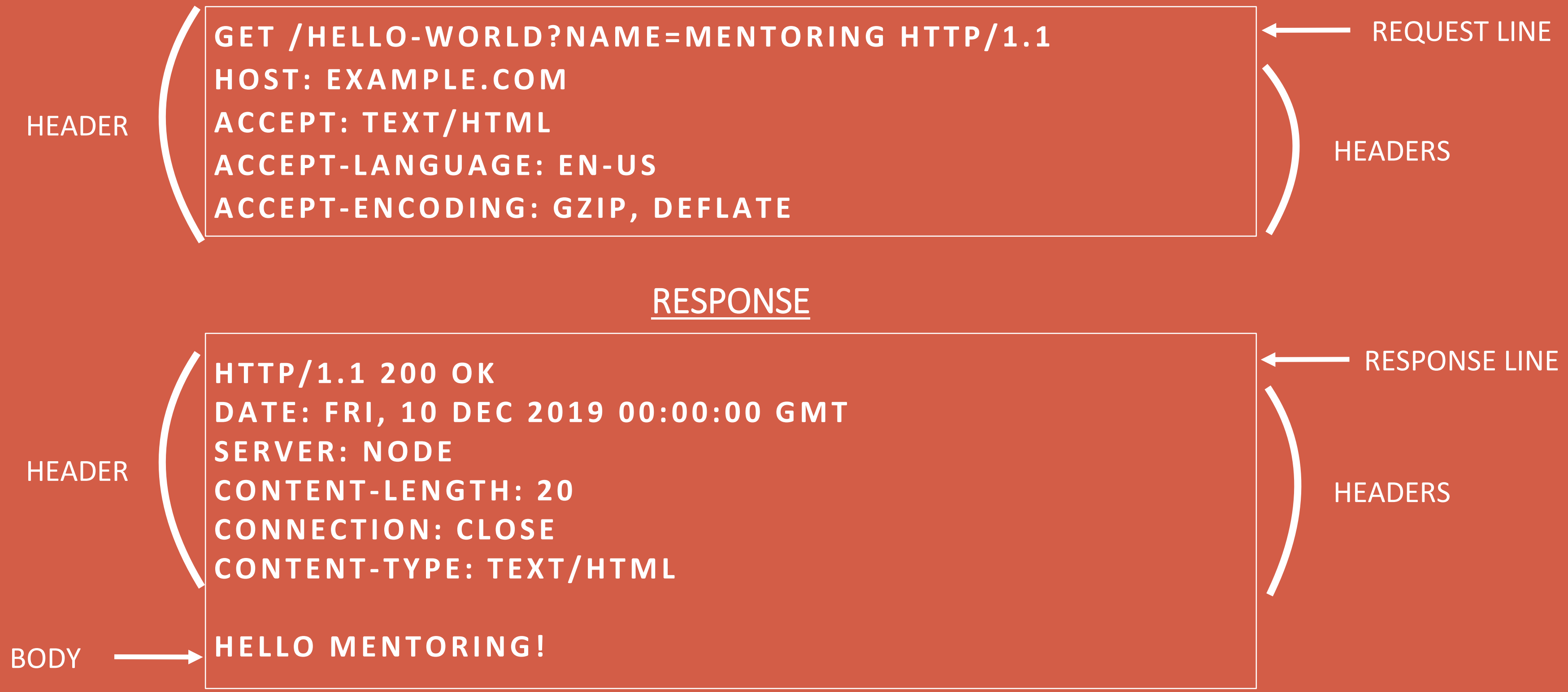
- BODY

The message (payload)

Is separated from the header with a blank line

It's optional both for Requests and Responses but remember to be polite

REQUEST



```
GET /HELLO-WORLD?NAME=MENTORING HTTP/1.1
HOST: EXAMPLE.COM
ACCEPT: TEXT/HTML
ACCEPT-LANGUAGE: EN-US
ACCEPT-ENCODING: GZIP, DEFLATE
```

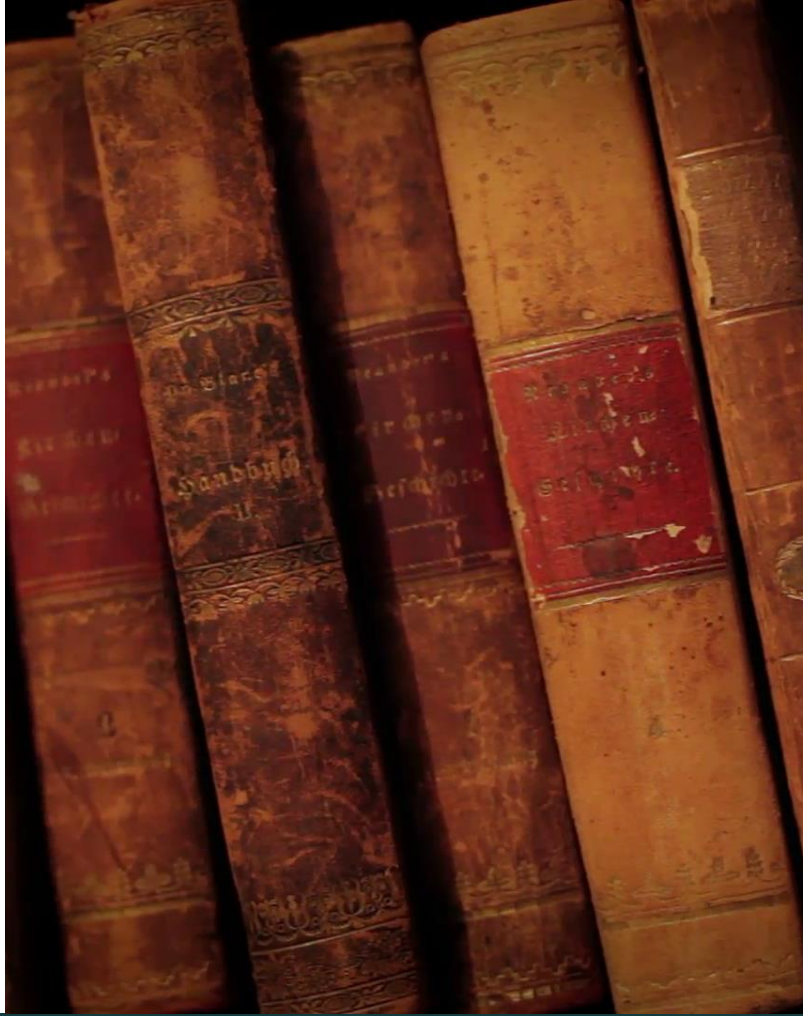
RESPONSE

```
HTTP/1.1 200 OK
DATE: FRI, 10 DEC 2019 00:00:00 GMT
SERVER: NODE
CONTENT-LENGTH: 20
CONNECTION: CLOSE
CONTENT-TYPE: TEXT/HTML

HELLO MENTORING!
```

Versions

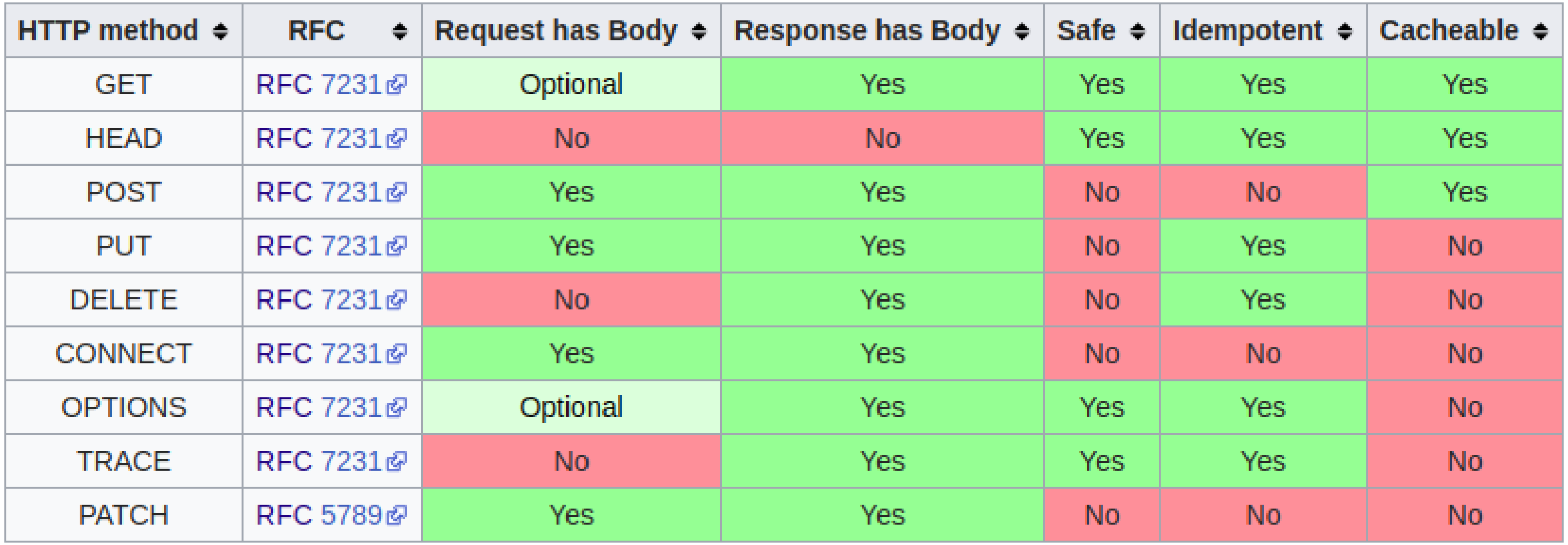
- **HTTP v0.9** (circa 1991)
- **HTTP v1** (circa 1996) ***RFC1945***
- **HTTP v1.1** (circa 1997) ***RFC7230-7235***
- **SPDY** (circa 2012) Google has it
- **ALPN** (ex-NPN) (circa 2014) ***RFC7301***
- **HTTP v2** (circa 2015) ***RFC7540***
- **HTTP v3** (circa 4030) ***draft-ietf-quic-http-20***

RFC = Request for Comments



HTTP Methods (Request Side)

Safe = intended only for information retrieval and should not change the state of the server
Idempotent = multiple identical requests should have the same effect as a single request
Cacheable = you should probably cache what you get

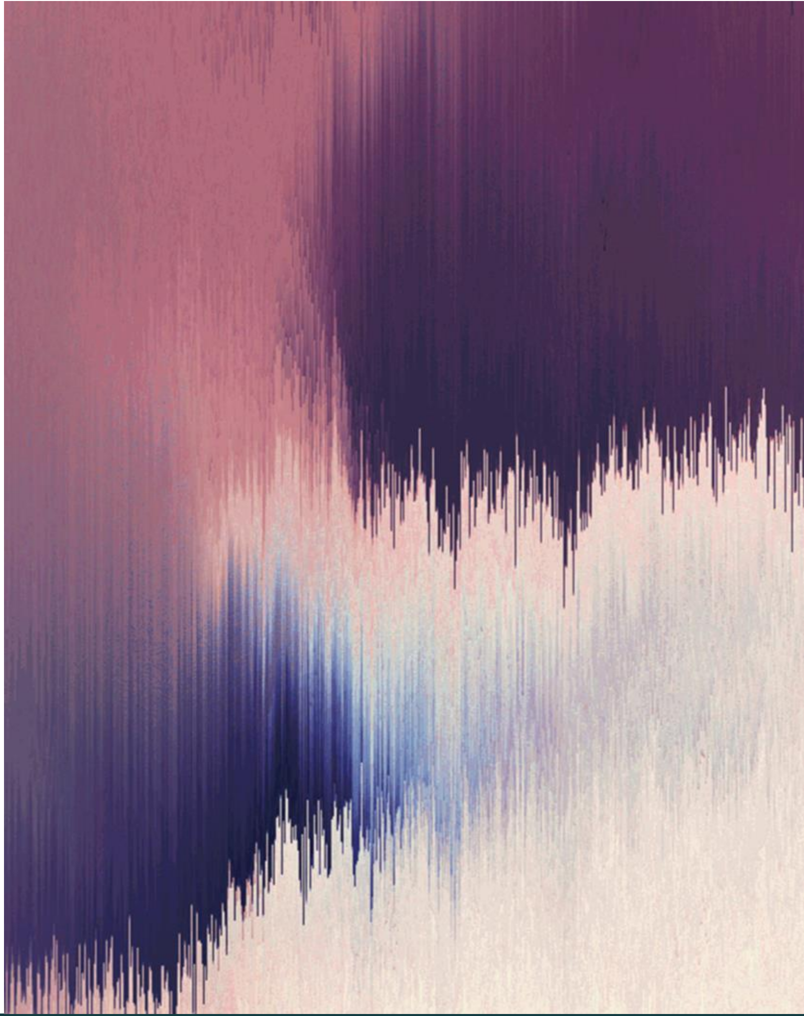


HTTP method	RFC	Request has Body	Response has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

RFC = Request for Comments

Status (Response Side)

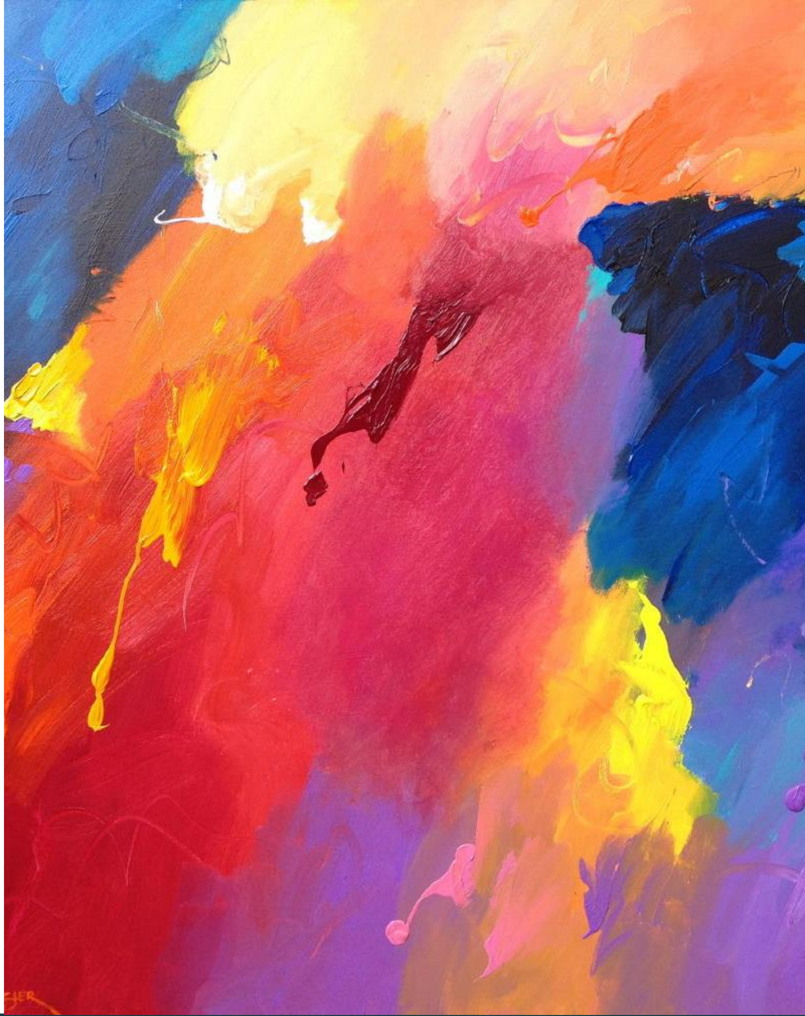
1xx	**INFORMATION**
2xx	**SUCCESS**
3xx	**REDIRECTION**
4xx	**CLIENT ERROR (REQUEST ERROR)**
5xx	**SERVER ERROR**



Headers

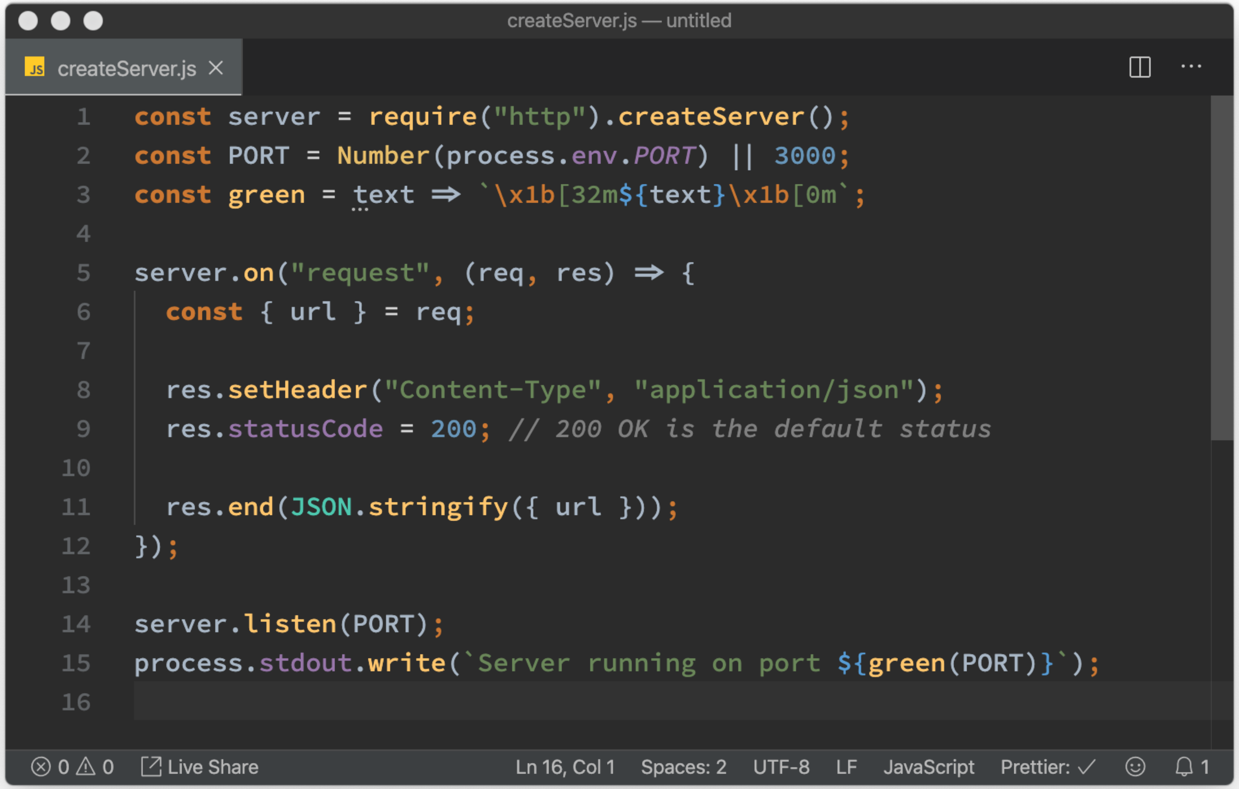
GROUPS

- Authentication
- Caching / Conditionals
- Client Hints / Server Hints
- Connection Management
- Content Negotiation / Message Body Information / Encoding
- Controls
- Cookies
- CORS
- Proxies
- Request Context / Response Context
- Security
- Etc



Node.js HTTP Module

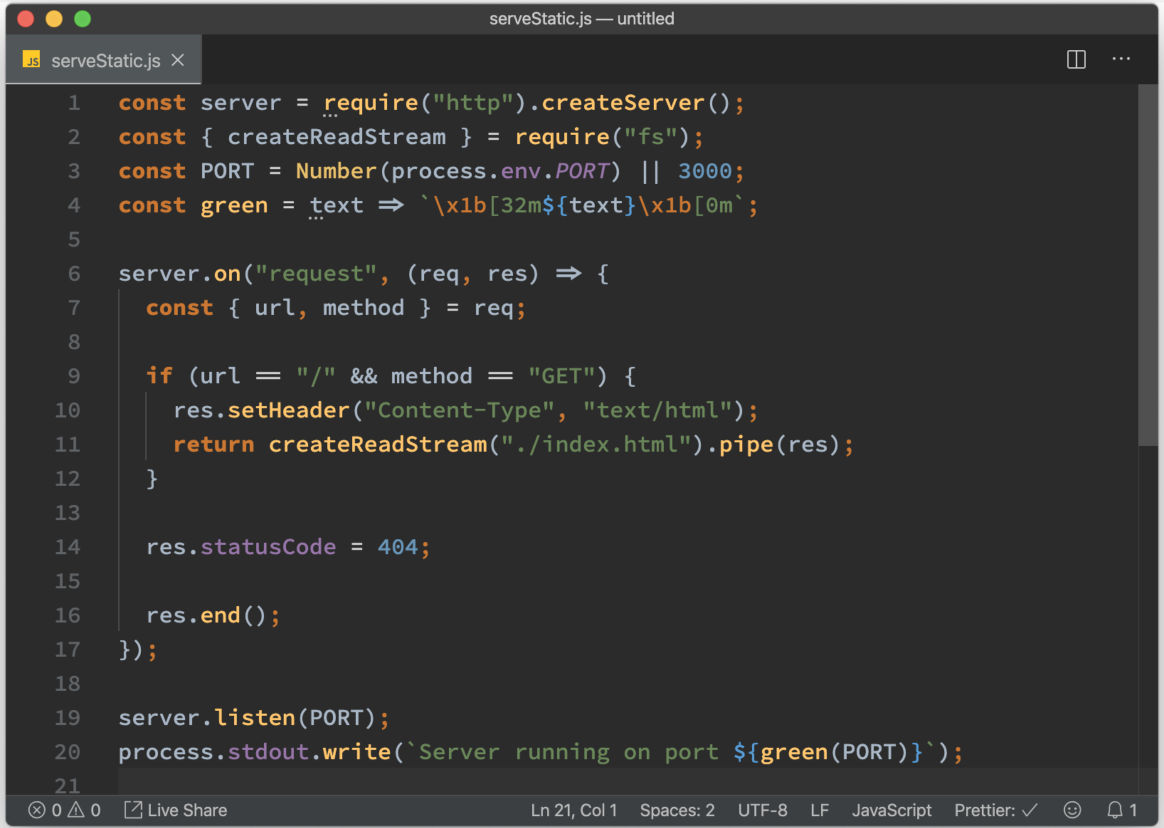
HTTP with Node.js


createServer.js — untitled

createServer.js

```javascript
const server = require("http").createServer();
const PORT = Number(process.env.PORT) || 3000;
const green = text => `\x1b[32m${text}\x1b[0m`;

server.on("request", (req, res) => {
  const { url } = req;

  res.setHeader("Content-Type", "application/json");
  res.statusCode = 200; // 200 OK is the default status

  res.end(JSON.stringify({ url }));
});

server.listen(PORT);
process.stdout.write(`Server running on port ${green(PORT)}`);

```

Serving static files with HTTP and Streams

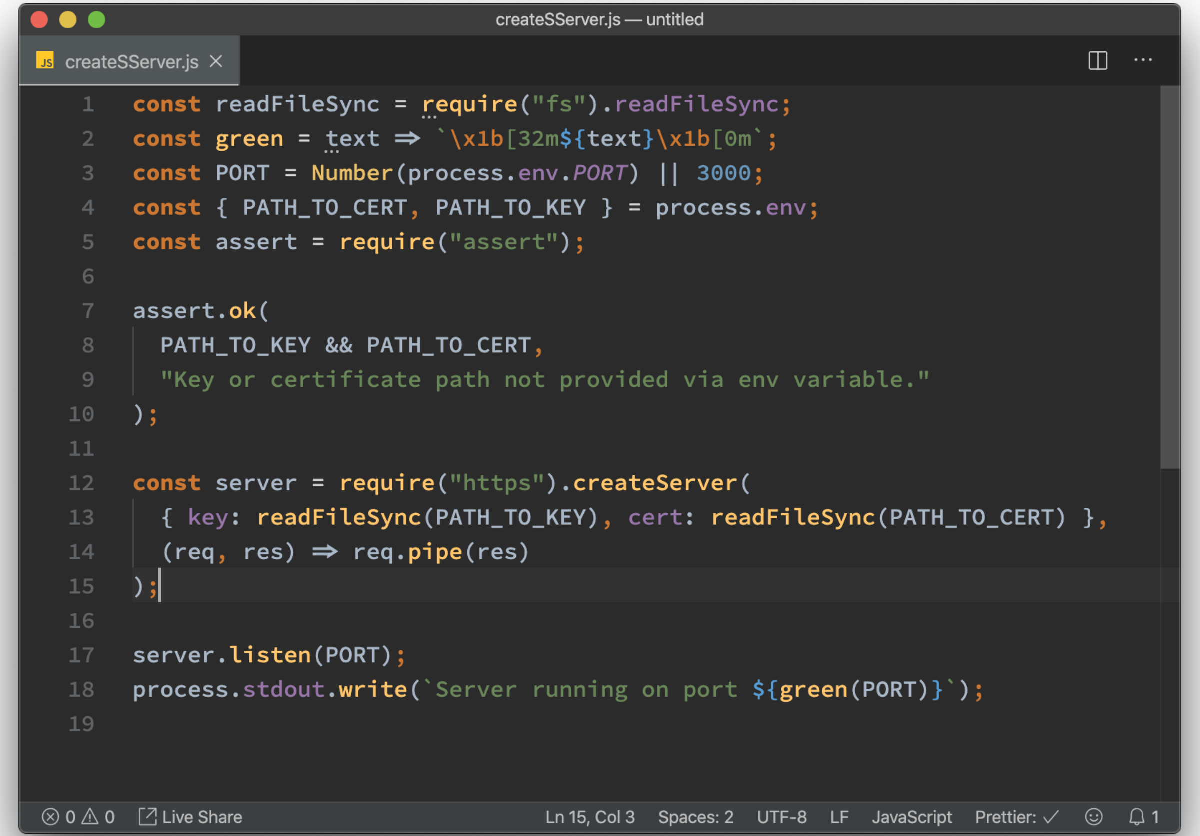


serveStatic.js — untitled

serveStatic.js

```javascript
const server = require("http").createServer();
const { createReadStream } = require("fs");
const PORT = Number(process.env.PORT) || 3000;
const green = text => `\x1b[32m${text}\x1b[0m`;

server.on("request", (req, res) => {
  const { url, method } = req;

  if (url == "/" && method == "GET") {
    res.setHeader("Content-Type", "text/html");
    return createReadStream("./index.html").pipe(res);
  }

  res.statusCode = 404;

  res.end();
});

server.listen(PORT);
process.stdout.write(`Server running on port ${green(PORT)}`);
```

HTTPS with Node.js

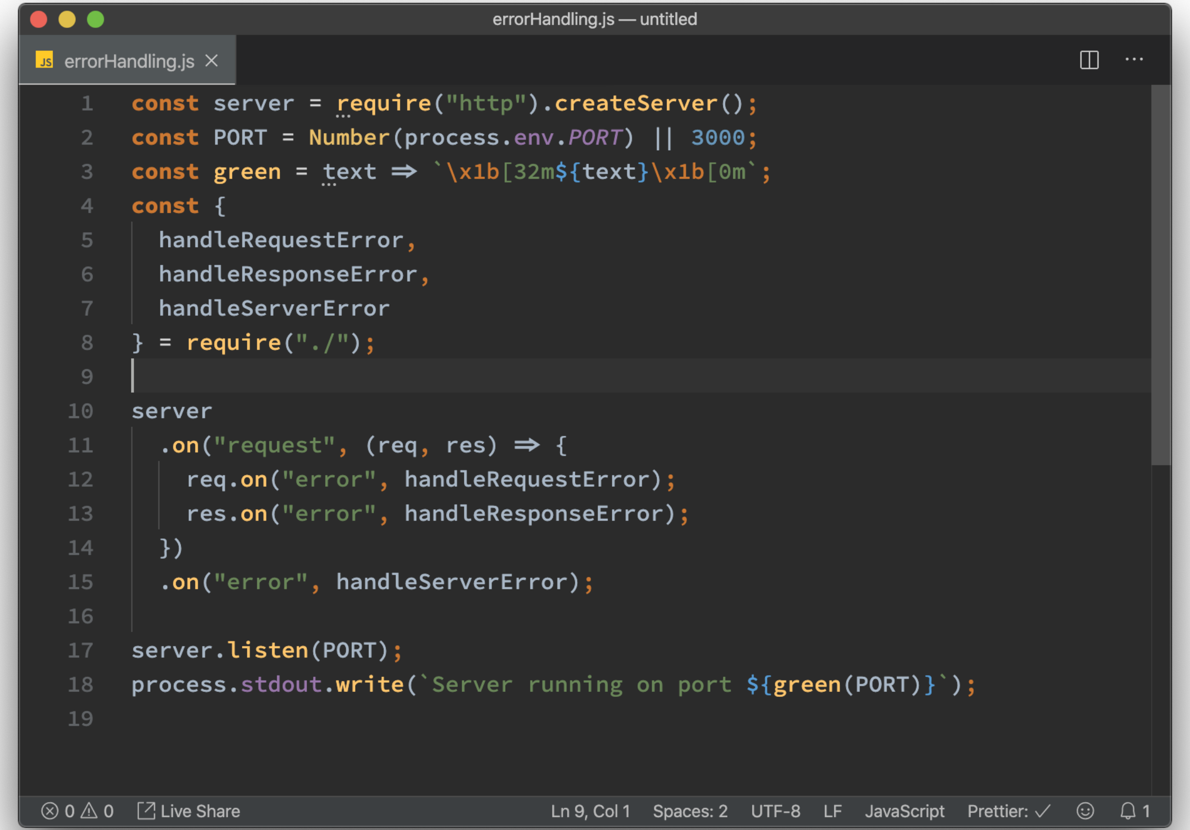


createSServer.js — untitled

```javascript
const readFileSync = require("fs").readFileSync;
const green = text => `\x1b[32m${text}\x1b[0m`;
const PORT = Number(process.env.PORT) || 3000;
const { PATH_TO_CERT, PATH_TO_KEY } = process.env;
const assert = require("assert");

assert.ok(
  PATH_TO_KEY && PATH_TO_CERT,
  "Key or certificate path not provided via env variable."
);

const server = require("https").createServer(
  { key: readFileSync(PATH_TO_KEY), cert: readFileSync(PATH_TO_CERT) },
  (req, res) => req.pipe(res)
);

server.listen(PORT);
process.stdout.write(`Server running on port ${green(PORT)}`);
```

Ln 15, Col 3   Spaces: 2   UTF-8   LF   JavaScript   Prettier: ✓

HTTP(s) Error handling



errorHandling.js — untitled

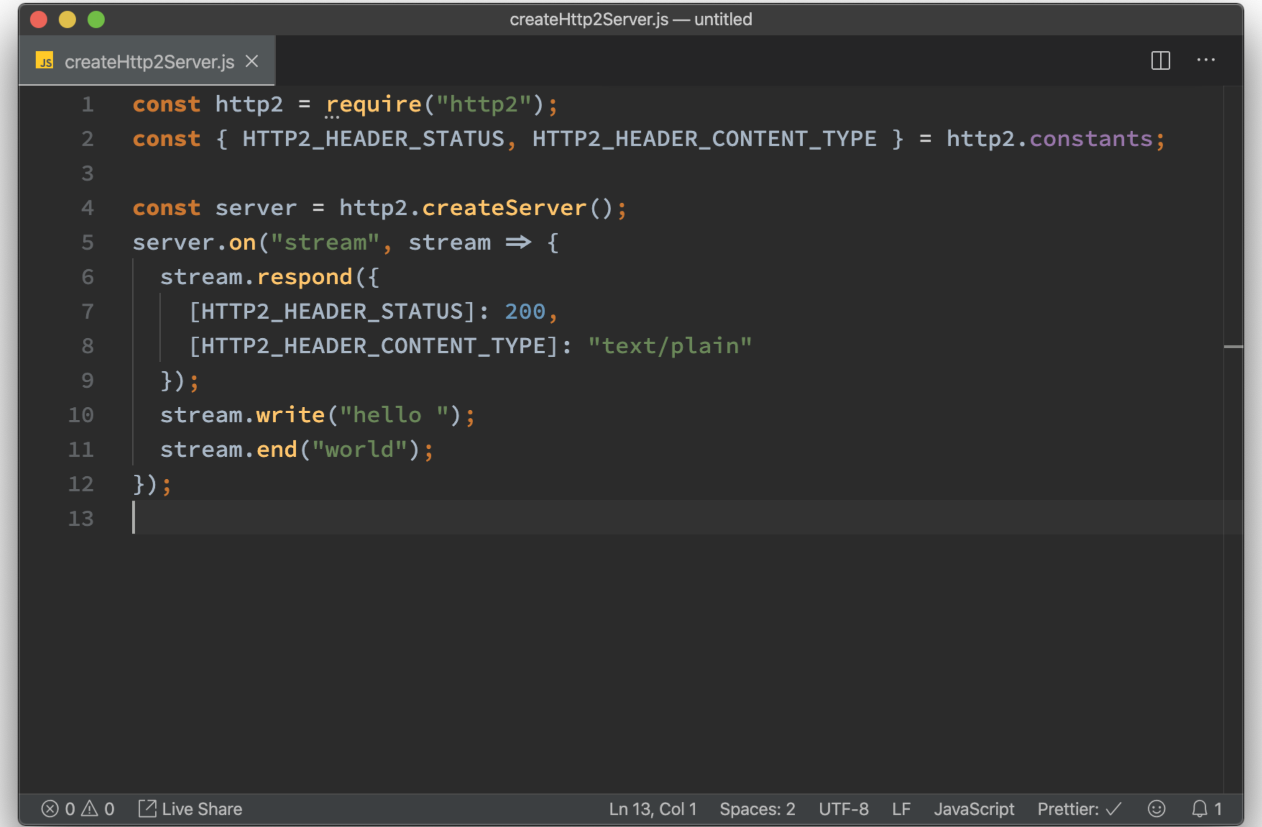
errorHandling.js

```javascript
const server = require("http").createServer();
const PORT = Number(process.env.PORT) || 3000;
const green = text => `\x1b[32m${text}\x1b[0m`;
const {
  handleRequestError,
  handleResponseError,
  handleServerError
} = require("./");

server
  .on("request", (req, res) => {
    req.on("error", handleRequestError);
    res.on("error", handleResponseError);
  })
  .on("error", handleServerError);

server.listen(PORT);
process.stdout.write(`Server running on port ${green(PORT)}`);

```

What’s up with HTTP2?

- It’s backwards compatible with HTTP/1.1
- It’s faster due to request headers compression and binary protocol
- It’s secure (you **MUST** encrypt)
- It supports **Server Push** (you can pass additional content alongside requested content if you feel like it) but you need to master it
- It preserves single connection
- It is slower than HTTP/1.1 with common front-end practices, e.g. sprites, file concatenation, etc.
- It is going to be updated to **HTTP/3** some time soon

HTTP/2 with Node.js


createHttp2Server.js — untitled
createHttp2Server.js

```javascript
const http2 = require("http2");
const { HTTP2_HEADER_STATUS, HTTP2_HEADER_CONTENT_TYPE } = http2.constants;

const server = http2.createServer();
server.on("stream", stream => {
  stream.respond({
    [HTTP2_HEADER_STATUS]: 200,
    [HTTP2_HEADER_CONTENT_TYPE]: "text/plain"
  });
  stream.write("hello ");
  stream.end("world");
});
```

⊗ 0 ⚠ 0 Live Share
Ln 13, Col 1 Spaces: 2 UTF-8 LF JavaScript Prettier: ✓ 1

Node.js and WebSocket Brief

REQUEST

REQUEST LINE

HEADER

```
GET /CHAT HTTP/1.1
HOST: SERVER.EXAMPLE.COM
UPGRADE: WEBSOCKET
CONNECTION: UPGRADE
SEC-WEBSOCKET-KEY: DGHLIHNHBXBSZSBUB25JZQ==
ORIGIN: HTTP://EXAMPLE.COM
SEC-WEBSOCKET-PROTOCOL: CHAT, SUPERCHAT
SEC-WEBSOCKET-VERSION: 13
```

RESPONSE

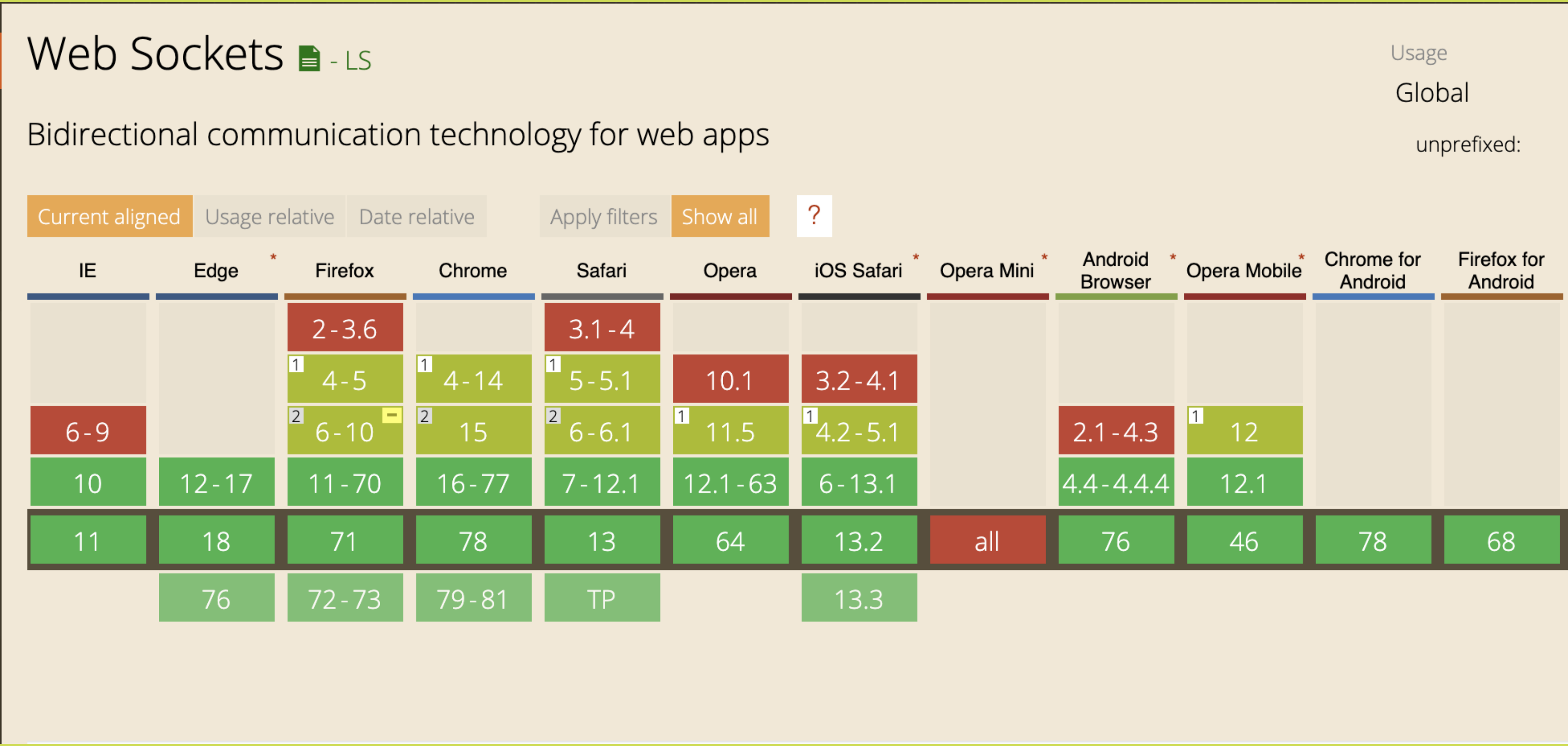
RESPONSE LINE

HEADER

```
HTTP/1.1 101 SWITCHING PROTOCOLS
UPGRADE: WEBSOCKET
CONNECTION: UPGRADE
SEC-WEBSOCKET-
ACCEPT: S3PPLMBITXAQ9KYGZZHZRBK+XOO=
SEC-WEBSOCKET-PROTOCOL: CHAT
```

WebSocket Overview

- Full-featured duplex stream
- WebSocket is a bidirectional communication protocol
- Connection can be closed from either side
- WebSocket is often used to handle real-time web applications
- Multiple Server and Client implementations in JavaScript
- Browser compatibility must be considered



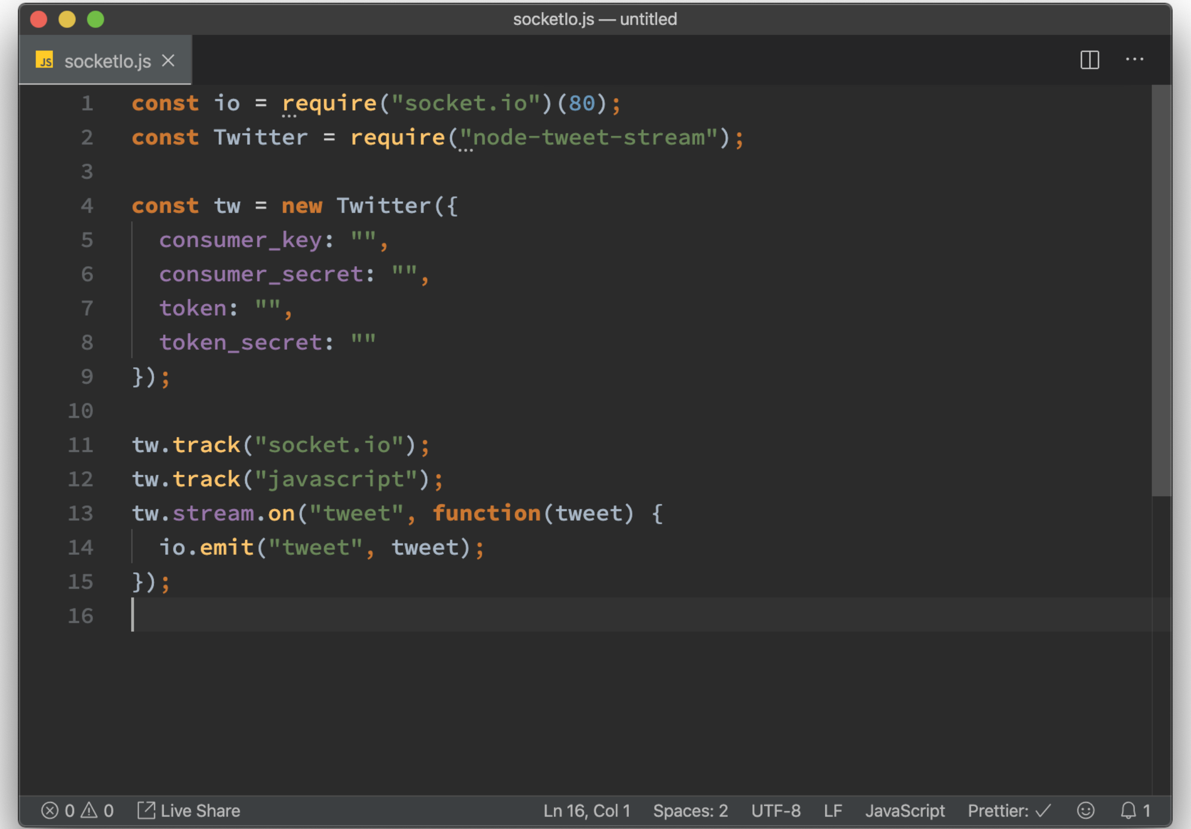
Web Sockets - LS

Usage
Global
unprefixed:

Bidirectional communication technology for web apps

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android
		2-3.6		3.1-4							
		[1] 4-5	[1] 4-14	[1] 5-5.1	10.1	3.2-4.1					
6-9		[2] 6-10	[2] 15	[2] 6-6.1	[1] 11.5	[1] 4.2-5.1		2.1-4.3	[1] 12		
10	12-17	11-70	16-77	7-12.1	12.1-63	6-13.1		4.4-4.4.4	12.1		
11	18	71	78	13	64	13.2	all	76	46	78	68
	76	72-73	79-81	TP		13.3					

Socket.io 2.0 Example


socketlo.js — untitled

socketlo.js

```
const io = require("socket.io")(80);
const Twitter = require("node-tweet-stream");

const tw = new Twitter({
  consumer_key: "",
  consumer_secret: "",
  token: "",
  token_secret: ""
});

tw.track("socket.io");
tw.track("javascript");
tw.stream.on("tweet", function(tweet) {
  io.emit("tweet", tweet);
});

```

References

- Anatomy of an HTTP Transaction
- HTTP. The Polite Parts (RU)
- WebSocket Protocol RFC
- Implementing a WebSocket Server with Node.js
- Socket.io
- Benchmark results for server-side JavaScript
- ES4x

That's all Folks!