NODE.JS

Part II

NEED FOR A TEMPLATE

❑If you want to add specific handling for different HTTP verbs (e.g. GET, POST, DELETE, etc.)

❑Separately handle requests at different URL paths ("routes")

❑ Serve static files, or use templates to dynamically create the response,

❑ Node won't be of much use on its own.

❑You will either need to write the code yourself, or you can avoid reinventing the wheel and use a web framework!

APPLICATION FRAMEWORKS

- provides frozen spots
  - overall architecture
  - How the components interact
- allows to concentrate in hot spots to extend the behaviour of the framework
  - Hot spots are the functions written for the application
- A framework is not suitable for a problem when …

WEB APPLICATION FRAMEWORKS

- An application framework that is designed to support development of web applications that generally includes
  - Database support
  - Templating framework for generating dynamic web content
  - HTTP session management with middleware support
  - Built-in testing framework
- It can also support internationalization, security and privacy
- Consistent look and feel and consistent with database

WEB FRAMEWORKS EXAMPLES

- Ruby on Rails
- Play
- ASP.NET
- Django
- Symfony
- Spring
- Vue.js
- Angular js

EXPRESS

❑ Express is the most popular *Node* web framework, and is the underlying library for a number of other popular Node web frameworks.

❑ It provides mechanisms to:

❑ Write handlers for requests with different HTTP verbs at different URL paths (routes).

❑ Integrate with "view" rendering engines in order to generate responses by inserting data into templates.

❑ Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.

❑ Add additional request processing "middleware" at any point within the request handling pipeline.

EXPRESS FEATURES

- ❑ *Express* itself is fairly minimalist
- ❑ Developers have created compatible middleware packages to address almost any web development problem
- ❑ There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and *many* more.
- ❑You can find a list of middleware packages maintained by the Express team at Express Middleware (along with a list of some popular 3rd party packages).

IS EXPRESS OPINIONATED

- ❑ Opinionated frameworks are those with opinions about the "right way" to handle any particular task. They often support rapid development *in a particular domain* (solving problems of a particular type) because the right way to do anything is usually well-understood and well-documented

- ❑ Unopinionated frameworks, by contrast, have far fewer restrictions on the best way to glue components together to achieve a goal, or even what components should be used.

EXPRESS OPINION

- ❑ Express is unopinionated.
- ❑ You can insert almost any compatible middleware you like into the request handling chain, in almost any order you like.
- ❑ You can structure the app in one file or multiple files, using any directory structure.
- ❑ You may sometimes feel that you have too many choices

EXPRESS FEATURES

❑Express provides methods to specify what function is called for a particular HTTP verb (GET, POST, SET, etc.) and URL pattern ("Route")

❑ Provides methods to specify what template ("view") engine is used

❑ Where template files are located

❑ What template to use to render a response

❑You can use Express middleware to add support for cookies, sessions, and users, getting POST/GET parameters, etc.

❑You can use any database mechanism supported by Node (Express does not define any database-related behavior).

EXPRESS APP

□This object, which is traditionally named app, has methods for

□ routing HTTP requests

□ configuring middleware,

□rendering HTML views,

□registering a template engine, and

□ modifying application settings that control how the application behaves (e.g. the environment mode, whether route definitions are case sensitive, etc.)

□res.json() to send a JSON response or res.sendFile() to send a file

```
const express = require("express");
const app = express();
const port = 3000;

app.get("/", function (req, res) {
  res.send("Hello World!");
});

app.listen(port, function () {
  console.log(`Example app listening on
port ${port}!`);
});
```

REQUEST HANDLING

A common convention for Node and Express is to use error-first callbacks. In this convention, the first value in your *callback functions* is an error value, while subsequent arguments contain success data.

❑ The *Express application* object also provides methods to define route handlers for all the other HTTP verbs, which are mostly used in exactly the same way:

❑checkout(), copy(), **delete()**, **get()**, head(), lock(), merge(), mkactivity(), mkcol(), notify(), options() patch(), **post()**, purge(), **put()**, report(), search(), subscribe(), trace(), unlock(), unsubscribe()

❑There is a special routing method, app.all(), which will be called in response to any HTTP method.

❑This is used for loading middleware functions at a particular path for all request methods.

```
app.all("/secret", function (req, res, next) {
         console.log("Accessing the secret section…");
          next(); // pass control to the next handler });
```

CREATING THE SERVER

```
const express = require('express’);
const http = require('http’);
const app = express();
const server = http.Server(app);
// Configuration
server.listen(process.env.PORT || 8000, () => {
console.log(`[ server.js ] Listening on port
${server.address().port}`);
 });
```

- Express allows us to create the webserver and REST API needed to run our app
  - Express is used here to create the http server and attaching it to an Express app
- Socket.io is the real-time engine (utilizing the WebSocket API)

ROUTING THROUGH EXPRESS

```
const wiki = require("./wiki.js");

app.use("/wiki", wiki);
```

```
// Home page route
router.get("/", function (req, res) {
  res.send("Wiki home page");
});
```

❑Often it is useful to group route handlers for a particular part of a site together and access them using a common route-prefix

❑a site with a Wiki might have all wiki-related routes in one file and have them accessed with a route prefix of */wiki/*

❑ In *Express* this is achieved by using the express.Router object.

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

MIDDLEWARE

- Middleware is used extensively in Express apps, for tasks from serving static files to error handling, to compressing HTTP responses

- Route functions end the HTTP request-response cycle by returning some response to the HTTP client, middleware functions *typically* perform some operation on the request or response and then call the next function in the "stack", which might be more middleware or a route handler

- The order in which middleware is called is up to the app developer.

- It is almost always the case that middleware is called before setting routes, or your route handlers will not have access to functionality added by your middleware.

- The **only** difference between a middleware function and a route handler callback is that middleware functions have a third argument next, which middleware functions are expected to call if they are not that which completes the request cycle

MIDDLEWARE IN EXPRESS

❑ To use third party middleware you first need to install it into your app using npm

```
    npm install morgan
```

❑ You could then call use() on the *Express application object* to add the middleware to the stack

```
// Function added with use() for all routes and verbs
app.use(a_middleware_function);
 // Function added with use() for a specific route
app.use("/someroute", a_middleware_function);
// A middleware function added for a specific HTTP verb and
route app.get("/", a_middleware_function);
```

CALLING MIDDLEWARE

```
const express = require("express");
const app = express();
const a_middleware_function = function (req, res, next) {
  // Perform some operations
  next(); // Call next() so Express will call the next
middleware function in the chain.
};
```

SERVING STATIC CONTENT

- ❑ Any files in the public directory are served by adding their filename (*relative* to the base "public" directory) to the base URL

```
app.use(express.static("public"));
```

- ❑ The URLS can be

```
 http://localhost:3000/images/dog.jpg
 http://localhost:3000/css/style.css
```

- ❑ If a file cannot be found by one middleware function then it will be passed on to the subsequent middleware (the order that middleware is called is based on your declaration order).

```
app.use(express.static(“media"));
```

- ❑ Hiding the path at the server

```
app.use("/media", express.static("public"));
```

ALL ABOUT ERRORS

- Errors are handled by one or more special middleware functions that have four arguments, instead of the usual three: (err, req, res, next)

```
app.use(function (err, req, res, next) {
     console.error(err.stack);
     res.status(500).send("Something broke!"); });
```

These can return any content required, but must be called after all other app.use() and routes calls so that they are the last middleware in the request handling process!

TEMPLATES

- Template engines (also referred to as "view engines") allow you to specify the *structure* of an output document in a template, using placeholders for data that will be filled in when a page is generated.

- Templates are often used to create HTML, but can also create other types of documents

- At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client

HTML TEMPLATE ENGINES

- Some popular template engines that work with Express are Pug, Mustache, and EJS.
- The Express application generator uses Jade as its default, but it also supports several others
- In your application settings code you set the template engine to use and the location where Express should look for templates using the 'views' and 'view engine' settings
- Directory to keep the template files

```
app.set('views', './views')
```

- To set pug as the template engine

```
app.set('view engine', 'pug’)
```

- The view engine cache does not cache the contents of the template’s output, only the underlying template itself. The view is still re-rendered with every request even when the cache is on.

PUG ENGINE

```
app.get("/views", function
(req, res) {
  res.send("Hello World!");
});
```

```
html
  head
    title= title
  body
    h1= message
    a(href = url) Homepage
```

```
app.get('/views', (req, res) => {
  res.render('index.pug', { title: 'Hey',
message: 'Hello there!' })
})
```

SETTING CONDITIONALS

```
html
    head
        title Simple template
    body
        if(user)
            h1 Hi, #{user.name}
        else
            a(href = "/wiki") Sign Up
```

```
router.get("/about", function (req, res) {
 res.render('signup.pug', {message: 'Wiki
home page',
url:"http://www.tutorialspoint.com"});
});
```

```
router.get("/about", function (req, res) {
 res.render('signup.pug', {{user: {name:
"Aritra", age: "22"}
});
```

ROUTING THROUGH THE FOLDERS

❑ In pug file insert the image source

```
img(src = '/testImage.jpg' alt = 'Testing Image’)
```

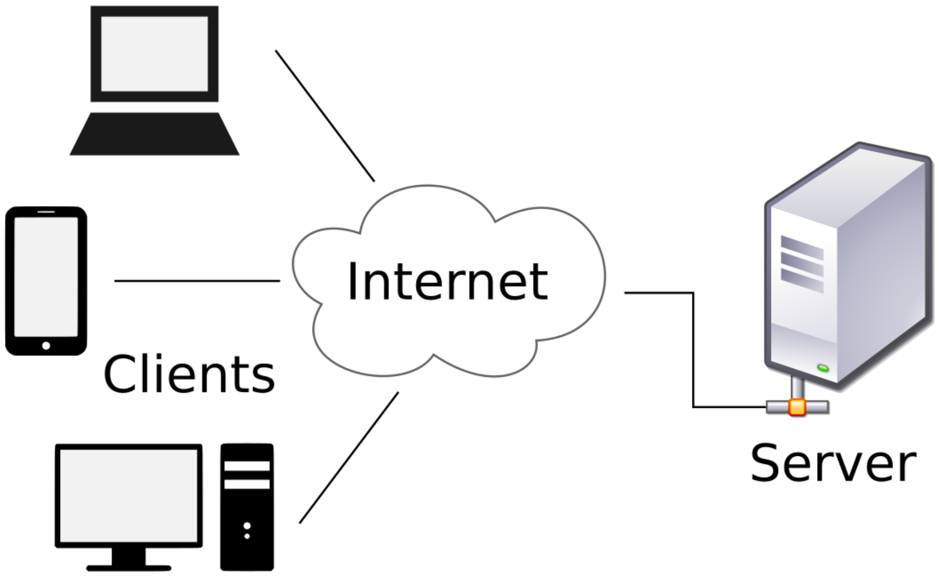
❑Insert static content

```
app.use(express.static('views’));
```

❑Adding a virtual path for the static content

```
app.use('/static', express.static('views'));
```

HTTP IS CLIENT DRIVEN

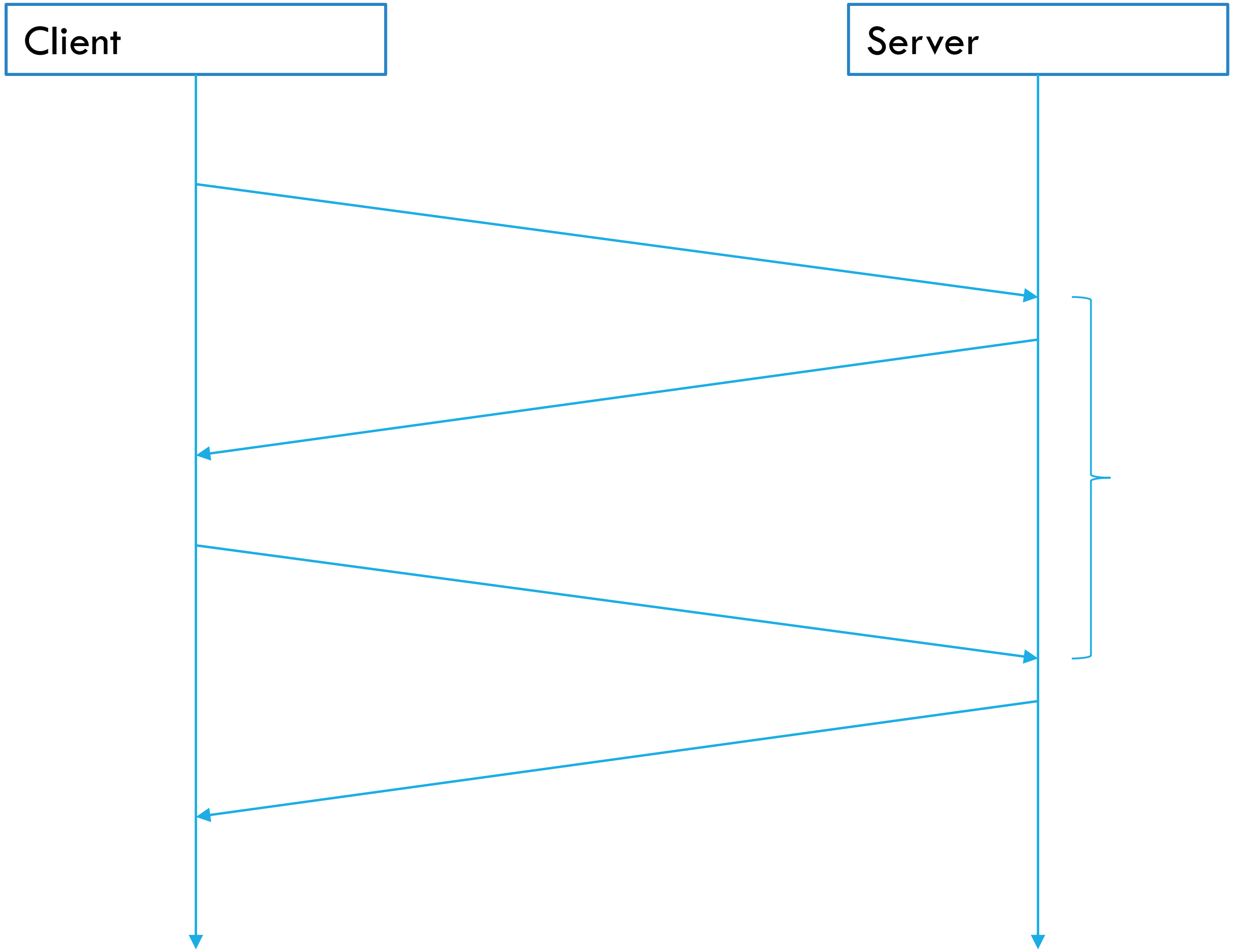


If client 2 updates important data that client 1 just pulled in, unless client 1 makes a second request for an update the server cannot notify client 1

Server is not able to push data as and when needed

PUSHING DATA

- ❑Manual- User may click on “refresh” when (s)he has decided to pull the data
- ❑Every time the window is opened data is pulled from the server
- ❑Periodic update
  - ❑Overhead on the server processes as it needs to process the request even when no update is to be notified
  - ❑Polling
  - ❑Exponential backoff



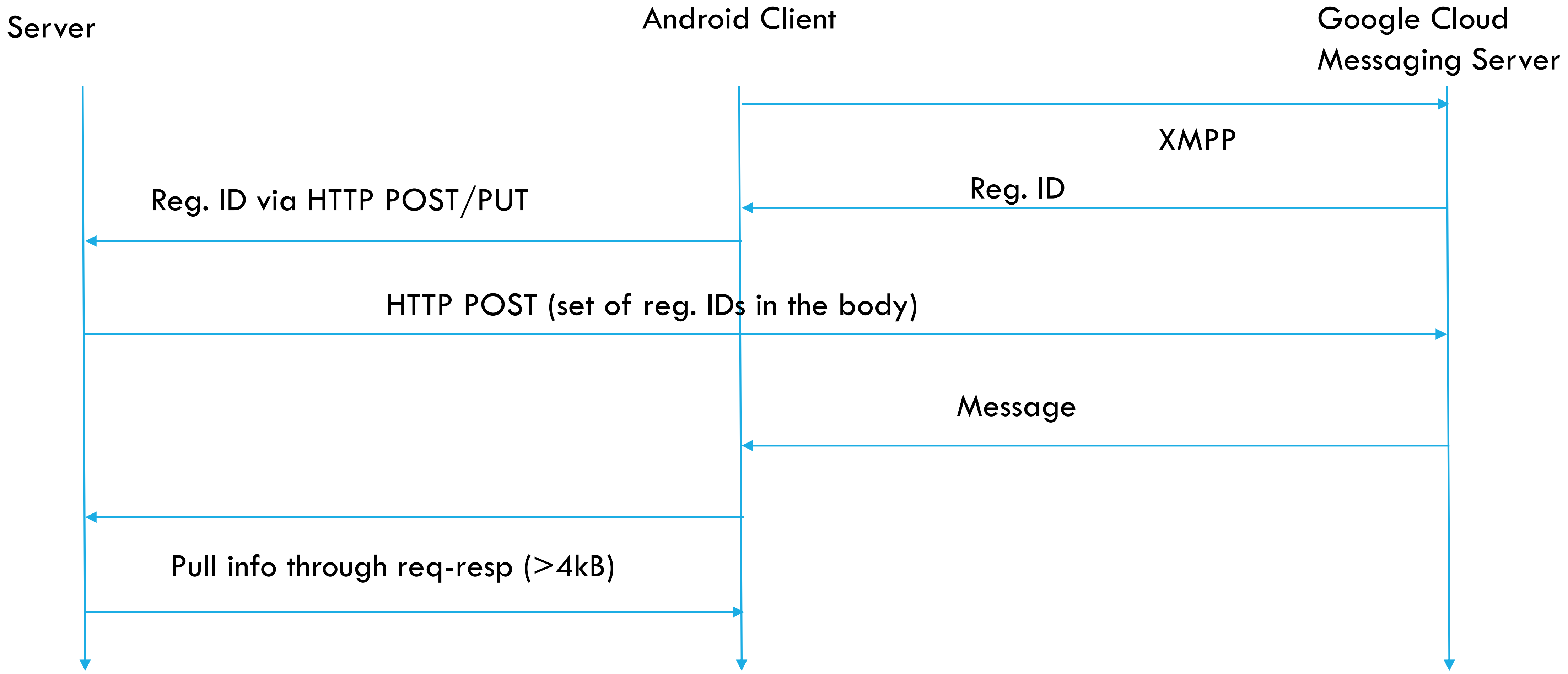
PERIODIC UPDATE

https://dzone.com/articles/thoughts-on-server-sent-events-http2-and-envoy-1

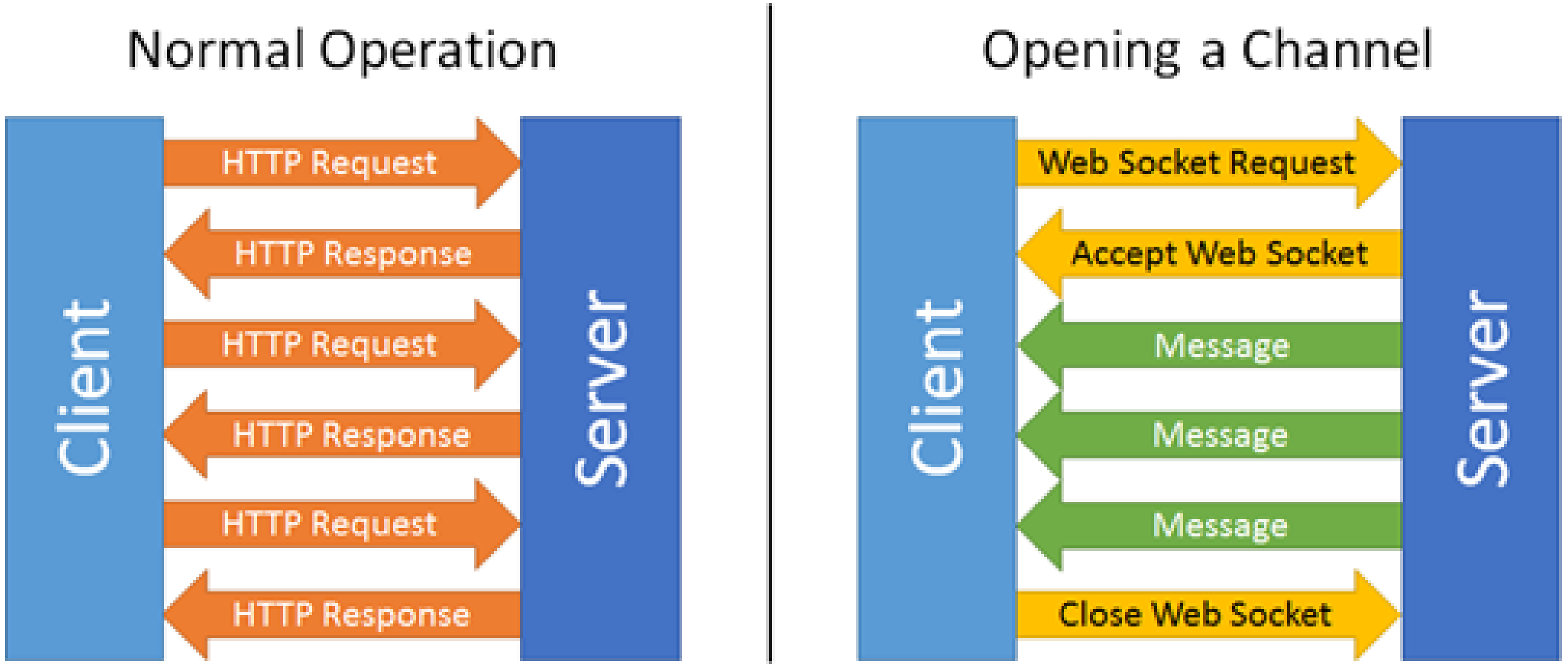
INTERMITTENT INTERNET CONNECTION

- How to handle intermittent internet connections
  - Event mechanisms and notifications should be designed to detect disconnection and reconnection
  - Periodically reconnect/ or getting an event from Android

PUSH NOTIFICATION

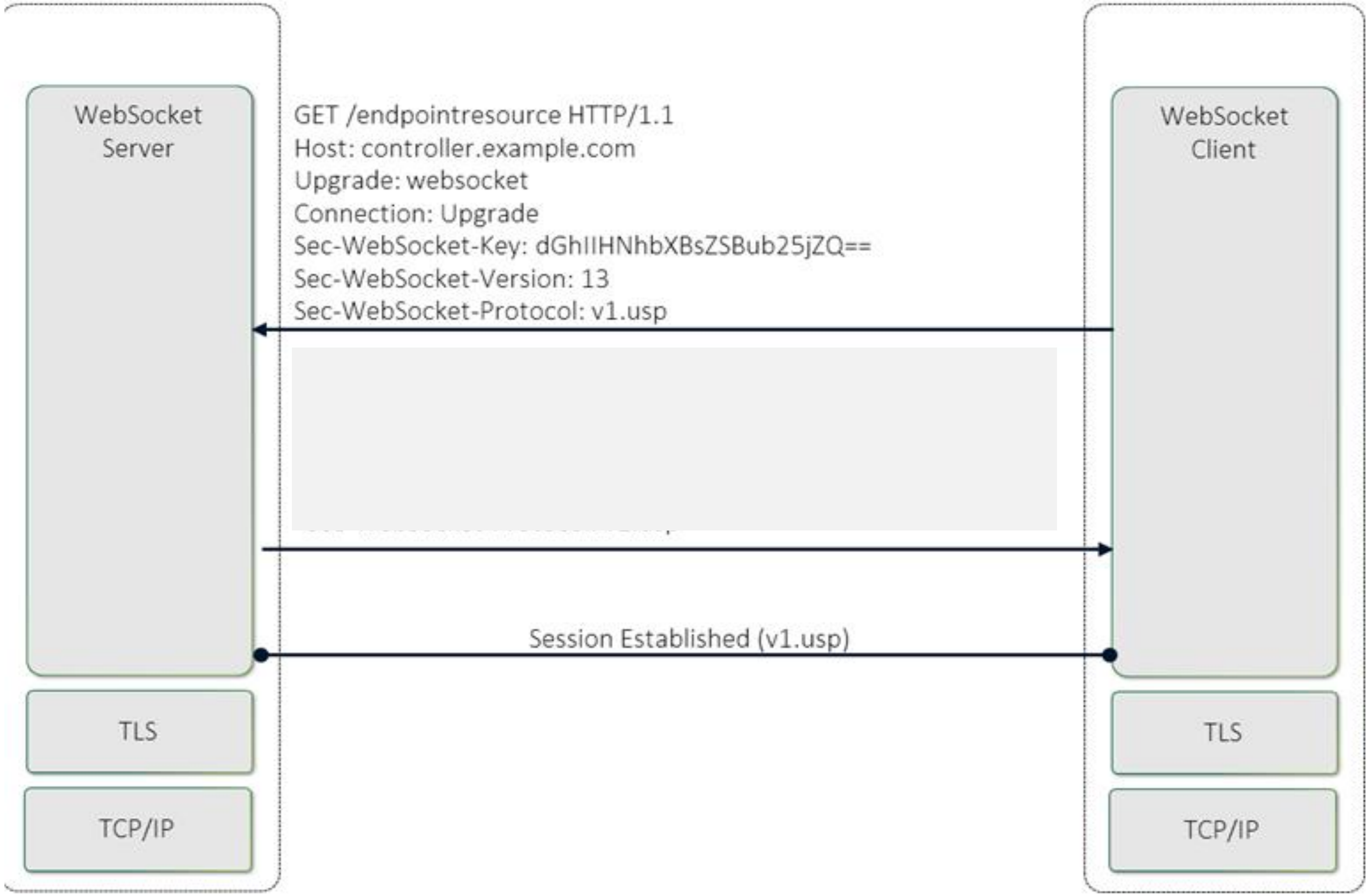

Server
Android Client
Google Cloud
Messaging Server
XMPP
Reg. ID
Reg. ID via HTTP POST/PUT
HTTP POST (set of reg. IDs in the body)
Message
Pull info through req-resp (>4kB)

WEBSOCKETS



- ❑communication protocol which features bi-directional, full-duplex communication over a persistent TCP connection
- ❑Any party can push data anytime
- ❑Single TCP connection for full duplex traffic
- ❑Message transfer on websockets does not require all parts of HTTP to be sent (header, URL, content type, body etc.)
- ❑Simply send binary messages or some other format back and forth in a server
- ❑By default, port 80 is used
- ❑Port 443 is used for connection tunneled over the TLS

WEB SOCKET HTTP COMPATIBILITY



```
GET /endpointresource HTTP/1.1
Host: controller.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Sec-WebSocket-Protocol: v1.usp
```

WEB SOCKET ADVANTAGES

❑ Stateful connection

❑ Message overhead of polling is more than web socket

❑ STOMP- Simple Text Oriented Messaging Protocol

https://spring.io/guides/gs/messaging-stomp-websocket/

S. El Mimouni and M. Bouhdadi, "Formal modeling of the Simple Text Oriented Messaging Protocol using Event-B method," 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), 2015, pp. 1-4, doi: 10.1109/AICCSA.2015.7507170.

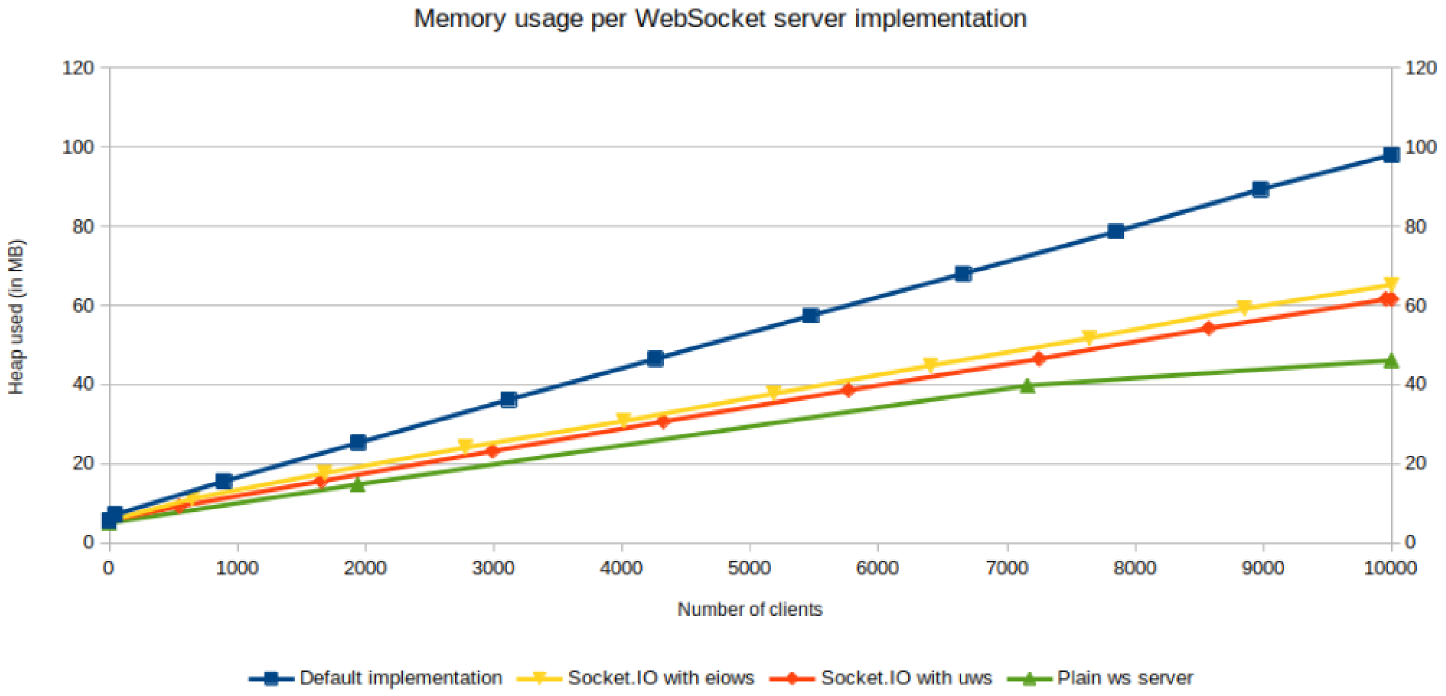
WEB SOCKET PROBLEMS

- Web sockets enable a server to push data only if the client is connected
- Web sockets are difficult to synchronize with more clients
- Keeping an open connection can have substantial resource impact
- For shared hosting servers, web socket is not a scalable option
- http responses can be cached by browser or by proxies
  - There is no such built-in mechanism for requests sent via webSockets

SOCKET.IO

- Socket.IO is a library that enables **low-latency**, **bidirectional** and **event-based** communication between a client and a server.
- The bidirectional channel between the Socket.IO server (Node.js) and the Socket.IO client (browser) is established with a WebSocket connection whenever possible, and will use HTTP long-polling as fallback.
- Although Socket.IO indeed uses WebSocket for transport when possible, it adds additional metadata to each packet. That is why a WebSocket client will not be able to successfully connect to a Socket.IO server

SOCKET.IO



- Under some particular conditions, the WebSocket connection between the server and the client can be interrupted with both sides being unaware of the broken state of the link.
- That's why Socket.IO includes a heartbeat mechanism, which periodically checks the status of the connection.
- And when the client eventually gets disconnected, it automatically reconnects with an exponential back-off delay, in order not to overwhelm the server.

WRITING A CHAT APPLICATION

- Create a package.json file to store all the dependency

```
{
"name": "socket-chat-example",
"version": "0.0.1",
"description": "my first socket.io app",
"dependencies": {}
}
```

- dependencies will be populated automatically in the json file and packages will be installed in a sub directory called node_modules
- `npm install express`
- sample index.js to create a server
- `npm install -g nodemon`

SOCKET.IO

❑Express initializes app to be a function handler that you can supply to an HTTP server

❑We define a route handler / that gets called when we hit our website home.

❑We make the http server listen on port 3000.

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);

app.get('/', (req, res) => {
  res.send('<h1>Hello world</h1>');
});

server.listen(3000, () => {
  console.log('listening on *:3000');
});
```

https://socket.io/get-started/chat#:~:text=The%20main%20idea%20behind%20Socket,binary%20data%20is%20supported%20too.

SOCKET.IO

```
 app.get('/', (req, res) => {
      res.sendFile(__dirname +
'/index.html');
});
```

```
<html>
<head>Add CSS</head>
<body>
      <ul id="messages"></ul>
      <form id="form" action="">
        <input id="input" autocomplete="off"
        /><button>Send</button>
      </form>
</body>
</html>
```

- ❑ Socket.IO is composed of two parts:
- ❑ A server that integrates with (or mounts on) the Node.JS HTTP Server socket.io
- ❑A client library that loads on the browser side socket.io-client

```
<script src="/socket.io/socket.io.js"></script>
<script>
var socket = io();
</script>
```

SOCKET.IO

A new instance of socket.io by passing the server (the HTTP server) object

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const { Server } = require("socket.io");
const io = new Server(server);
```

❑To indicate connected users the index.js is updated as follows.

```
io.on('connection', (socket) => {
    console.log('a user connected');
});
```

CLIENT AND SERVER

```
<script src="/socket.io/socket.io.js"></script>
<script>
var socket = io();
</script>
```

❑ index.html should be modified to include the io client

❑ If you would like to use the local version of the client-side JS file, you can find it at node_modules/socket.io/client-dist/socket.io.js.

❑ No URLs are specified when io() is called, since it defaults to trying to connect to the host that serves the page

❑ loading socket.io client from content delivery network

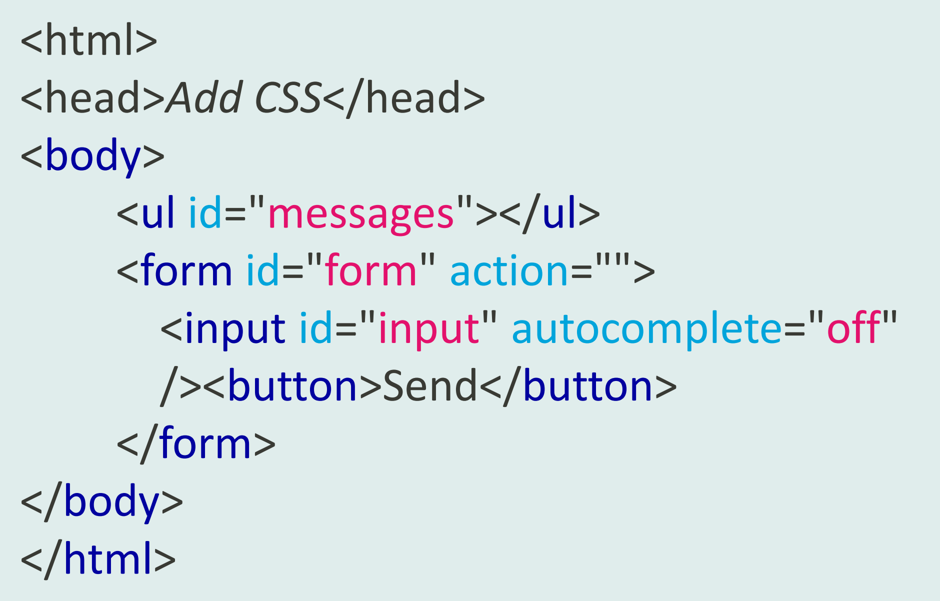
❑`<script src="https://cdn.socket.io/4.5.0/socket.io.min.js" integrity=“…" crossorigin="anonymous"></script>`

❑ events can be added for individual sockets

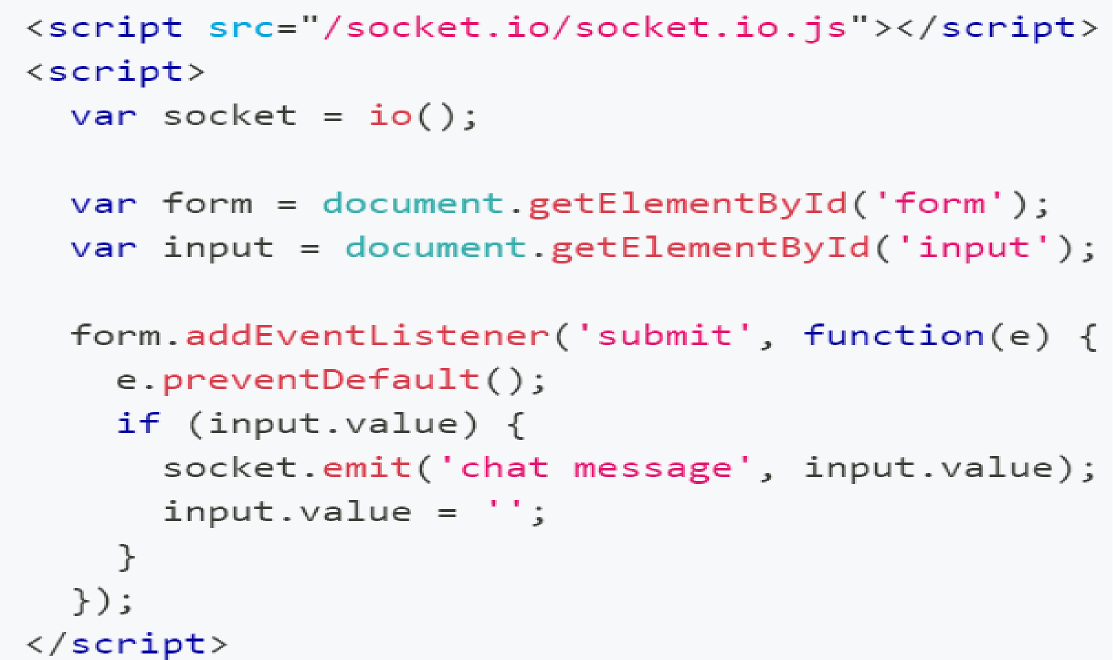
```
io.on('connection', (socket) => {
console.log('a user connected');
socket.on('disconnect', () => {
console.log('user disconnected');
});
});
```

CLIENT SENDS DATA

- ❑ The main idea behind Socket.IO is that you can send and receive any events you want, with any data you want. Any objects that can be encoded as JSON will do, and binary data is supported too.
- ❑ Let’s make it so that when the user types in a message, the server gets it as a chat message event



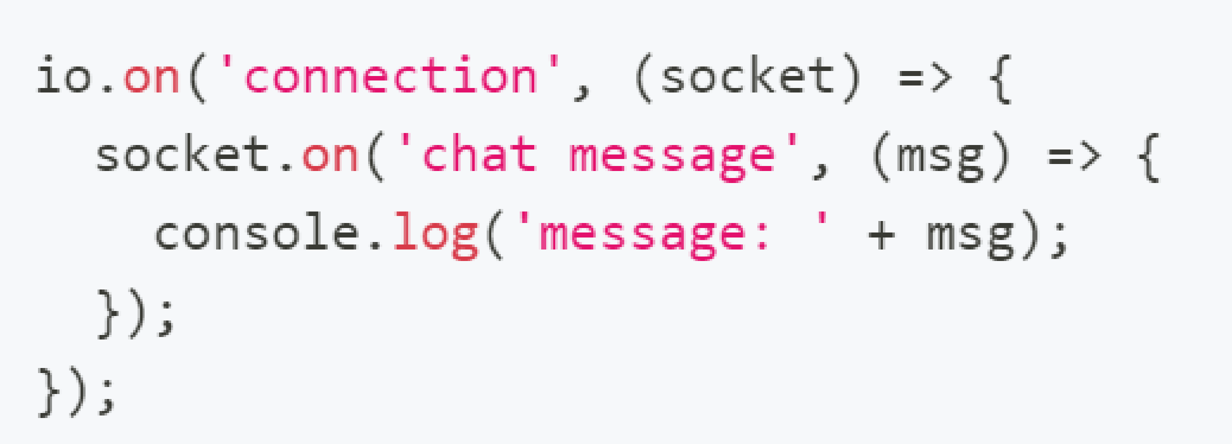
```
<html>
<head>Add CSS</head>
<body>
      <ul id="messages"></ul>
      <form id="form" action="">
         <input id="input" autocomplete="off"
         /><button>Send</button>
      </form>
</body>
</html>
```



```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();

  var form = document.getElementById('form');
  var input = document.getElementById('input');

  form.addEventListener('submit', function(e) {
    e.preventDefault();
    if (input.value) {
      socket.emit('chat message', input.value);
      input.value = '';
    }
  });
</script>
```



```
io.on('connection', (socket) => {
  socket.on('chat message', (msg) => {
    console.log('message: ' + msg);
  });
});
```

});

BROADCAST A MESSAGE

```
 io.emit('some event', { someProperty: 'some value',
otherProperty: 'other value' });
```

❑ Broadcasts a data to everyone including the sender

❑ If you want to send a message to everyone except for a certain emitting socket, we have the broadcast flag for emitting from that socket:

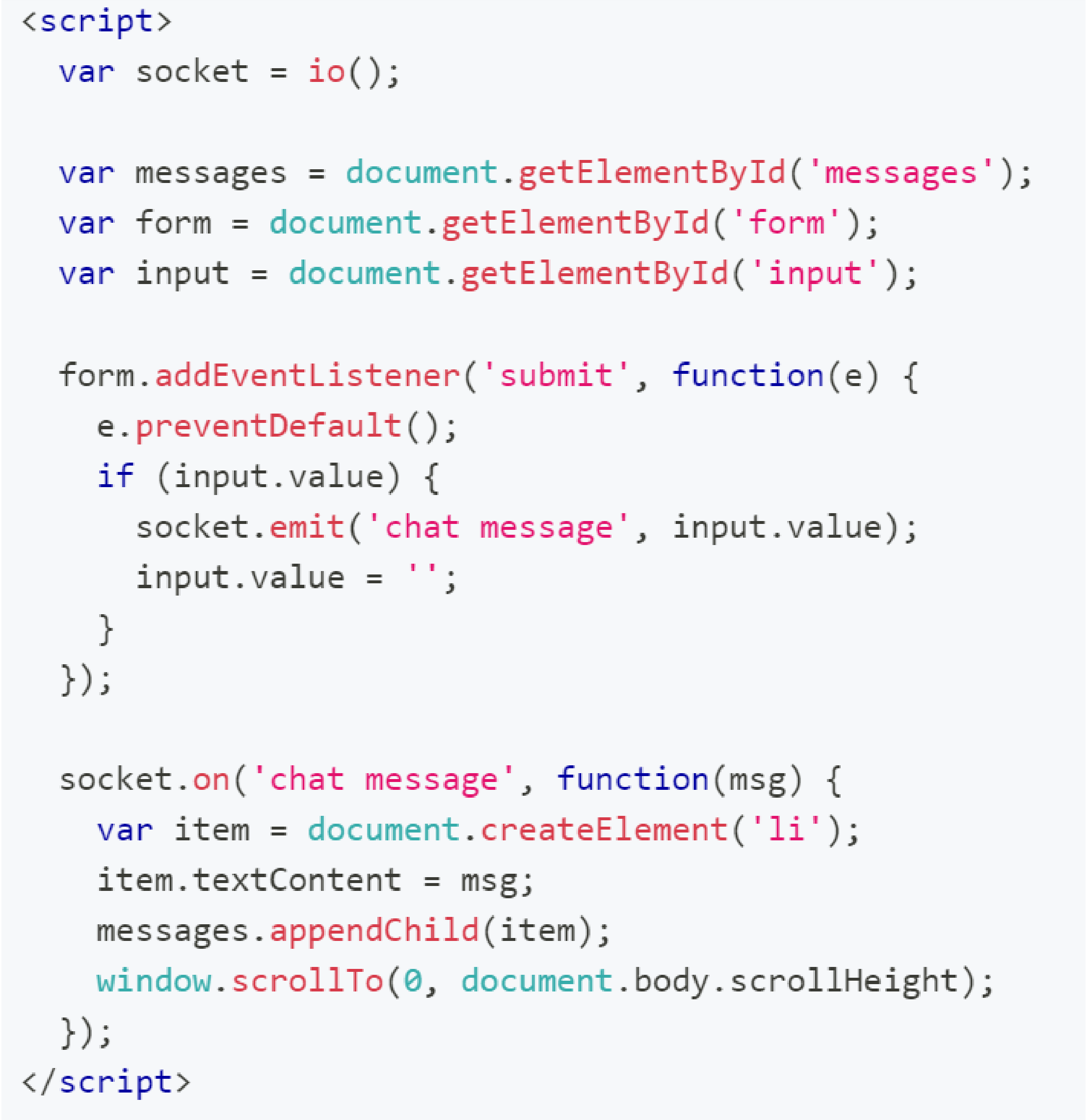
```
io.on('connection', (socket) => {
socket.broadcast.emit('hi’);
```

To broadcast the received message, at index.js the following is included

```
io.on('connection', (socket) => {
  socket.on('chat message', (msg) => {
    io.emit('chat message', msg);
  });
});
```

RECEIVING MESSAGE FROM OTHERS

on the client side when we capture a chat message event we’ll include it in the page



```
<script>
  var socket = io();

  var messages = document.getElementById('messages');
  var form = document.getElementById('form');
  var input = document.getElementById('input');

  form.addEventListener('submit', function(e) {
    e.preventDefault();
    if (input.value) {
      socket.emit('chat message', input.value);
      input.value = '';
    }
  });

  socket.on('chat message', function(msg) {
    var item = document.createElement('li');
    item.textContent = msg;
    messages.appendChild(item);
    window.scrollTo(0, document.body.scrollHeight);
  });
</script>
```

https://socket.io/get-started/chat