**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Socket.io\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Socket.io is a Javascript library that enables real-time, bidirectional communication between web clients(browsers) and servers. It simplifies the process of building interactive web applications by allowing data to be transmitted instantly and continuously the client and server, facilitating features like chat applications, live updates, and multiplayer gaming.

* **Bidirectional Communication?**

Bidirectional communication means that information can flow in two directions, both the web client (browser) and the server can send and receive data to and from each other in real-time. So, if something changes on the server, it can immediately notify the client, and vice versa, without the client needing to constantly ask the server if there’s new information available.

* **Course setup**
* npm i express nodemon socket.io
* add .gitignore
* write script -> start (package.json)
* add index.js and index.html
* **Basic code**

//1. packages

import express from 'express'

import http from 'http'

import { fileURLToPath } from 'node:url'

import { dirname, join } from 'node:path'

import { Server } from 'socket.io'

//2. instances

const app = express()

const server = http.createServer(app)

const io = new Server(server)

//3. Serving HTML file

const \_dirname = dirname(fileURLToPath(import.meta.url))

app.get('/', (req, res) => res.sendFile(join(\_dirname, "index.html")))

//4. Define a connection event handler

//5. Start the server

const PORT = 3000;

server.listen(PORT, ()=>console.log("server running on 3000"))

* **Connection (server and client)**
* **io.on(‘connection’, callback)**

io.on is a method used to register event listeners for different events that accur on the server-side. The object represents the main Socket.io server instance.

* When we call **io.on(‘connection’, callback),** we’re telling Socket.io to listen for a connection event, which occurs whenever a new client establishes a connection with the server. The callback function will be called/invoked whenever this event occurs, and it will receive a socket object representing the connection to the client.
* Similarly, we can use socket.on on the socket object to listen for events specific to that individual client connection.

//1. packages

import express from 'express'

import http from 'http'

import { fileURLToPath } from 'node:url'

import { dirname, join } from 'node:path'

import { Server } from 'socket.io'

//2. instances

const app = express()

const server = http.createServer(app)

const io = new Server(server)

//3. Serving HTML file

const \_dirname = dirname(fileURLToPath(import.meta.url))

app.get('/', (req, res) => res.sendFile(join(\_dirname, "index.html")))

//4. Define a connection event handler

io.on('connection', (socket)=>{

  console.log("User connected to server")

  // console.log(socket)

});

//5. Start the server

const PORT = 3000;

server.listen(PORT, ()=>console.log("server running on 3000"))

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Socket.io demo</title>

</head>

<body>

  <h1>Socket.io</h1>

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

  <script type="module">

    const socket = io();

    socket.on('connect', ()=>{

      console.log('Connected to client')

    })

  </script>

</body>

</html>

* **Disconnect (client and server)**

**Disconnection from the server side**

//4. Define a connection event handler

io.on('connection', (client)=>{

  console.log("User connected to server")

  client.on('disconnect', ()=>{

    console.log('User disconnected from the server')

  })

});

* We are not disconnecting the entire server, we just disconnecting the specific client from the server.

**Disconnection from the client side**

<script type="module">

    const socket = io();

    socket.on('connect', ()=>{

      console.log('Connected to client')

      //Disconnect the client from the server after 5 seconds

      setTimeout(()=>{

        socket.disconnect()

      }, 5000)

    })

    socket.on('disconnect', ()=>{

      console.log('Disconnected from the client')

    })

  </script>

* **Emitting Events(Theory)**

Emitting events allows us to send the data from one side (client or server) to the other. It’s a fundamental feature of real-time communication.

* **Emitting Events(Code)**
* Server to the client

Server(index.js)

io.on('connection', (client)=>{

  console.log("User connected to server")

  //Emit a 'message' event from server side to the client side

  client.emit('message',"Server is sending the data to the client")

// get a message from the client

client.on('new message', (message)=>{

    console.log(message)

  })

  client.on('disconnect', ()=>{

    console.log('User disconnected from the server')

  })

});

Client (index.html)

<script type="module">

    const socket = io();

    socket.on('connect', () => {

      console.log('Connected to client')

    })

    //get the data from the server

    socket.on('message' ,(message)=>{

      console.log(message)

    })

//get Emit a 'message' event from client side to the server side

    socket.emit("message", "server is sending that data to the client")

    socket.on('disconnect', () => {

      console.log('Disconnected from the client')

    })

  </script>

* **Basic Chat App.**