https://shorturl.at/kAHNV

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Socket io Chat application**

**API gateways**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

================================================================

Socket.io (Chat application)

================================================================

**Server-side (Node.js) Code:**

- It imports the required modules, such as express for the web server and socket.io for handling WebSocket connections.

- Creates an Express application instance and sets up a static directory to serve static files (like HTML, CSS, JS) from a folder named 'public'.

- Defines a port for the server to listen on, either from the environment variable PORT or defaults to 8080.

- Starts the server and listens on the specified port.

- Sets up a WebSocket connection using Socket.IO.

- Handles various events like connection, disconnection, and message sending.

**WebSocket Handling (Socket.IO):**

- io.on('connection', onConnected): This listens for any new WebSocket connection and calls the onConnected function when a client connects.

- onConnected(socket): This function handles events specific to each connected client (socket).

o Logs the connection.

o Adds the socket ID to a set to keep track of connected clients.

o Emits the current number of connected clients to all clients.

o Listens for disconnection events and updates the number of connected clients accordingly.

o Broadcasts incoming messages to all connected clients except the sender.

o Handles typing feedback by broadcasting the status of typing to all other clients.

- The server-side Socket.IO setup thus handles the core functionality of managing client connections, message broadcasting, and feedback.

**Client-side (Frontend) Code:**

- It retrieves necessary HTML elements for displaying chat-related information.

- Listens for form submission events (for sending messages), focus, keypress, and blur events on the message input field to handle typing feedback.

- Sends messages to the server upon form submission and emits typing feedback to the server upon focus, keypress, and blur events.

- Receives messages and typing feedback from the server and updates the UI accordingly.

Directory Structure

<>

public

-index.html

-script.js

-style.css

-Notification.mp3

-server.js

\*\*\*server.js\*\*\*

//import express module

const express = require('express')

//import path

const path = require('path')

//create rest object

const app = express()

//create port

const PORT = process.env.PORT || 8080

//start server

const server = app.listen(PORT, () => console.log(`server listening port ${PORT}`))

//use public directory static path for index.html

app.use(express.static(path.join(\_\_dirname, 'public')))

////////////////////////

//import socket io and pass server to it

const io = require('socket.io')(server)

//use set to maintain unique

let socketsConected = new Set()

//listen connection event

io.on('connection', onConnected)

function onConnected(socket) {

//handle connections

console.log('Socket connected', socket.id)

socketsConected.add(socket.id)

io.emit('clients-total', socketsConected.size)

//handle disconnections

socket.on('disconnect', () => {

console.log('Socket disconnected', socket.id)

socketsConected.delete(socket.id)

io.emit('clients-total', socketsConected.size)

})

//broadcast messages

socket.on('message', (data) => {

console.log(data)

socket.broadcast.emit('chat-message', data)

})

//typing status (feedback)

socket.on('feedback', (data) => {

socket.broadcast.emit('feedback', data)

})

}

\*\*\*index.html\*\*\*

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.14.0/css/all.min.css" />

<link rel="stylesheet" href="style.css" />

<title>Chat app</title>

</head>

<body>

<h1 class="title">Socketio Chat application</h1>

<div class="main">

<div class="name">

<span><i class="far fa-user"></i></span>

<input type="text" id="name-input" class="name-input" value="User" maxlength="20" />

</div>

<ul class="message-container" id="message-container">

<!-- These li elements are only for reference, and therefore, they are commented out... -->

<!--<li class="message-left">

<p class="message">

lorem impsun

<span>lata ● 26 July 10:40</span>

</p>

</li>

<li class="message-right">

<p class="message">

lorem impsun

<span>user ● 26 July 10:40</span>

</p>

</li>

<li class="message-feedback">

<p class="feedback" id="feedback">Mahesh is typing a message...</p>

</li>-->

</ul>

<form class="message-form" id="message-form">

<input type="text" name="message" id="message-input" class="message-input" />

<div class="v-divider"></div>

<button type="submit" class="send-button">

send <span><i class="fas fa-paper-plane"></i></span>

</button>

</form>

</div>

<h3 class="clients-total" id="client-total">Total clients: 2</h3>

<script src="/socket.io/socket.io.js"></script> <!-- this script is exposed when we are importing socket.io-->

<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.28.0/moment.min.js"></script>

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

</body>

</html>

\*\*\*script.js\*\*\*

//create connection for web socket

const socket = io()

const clientsTotal = document.getElementById('client-total')

const messageContainer = document.getElementById('message-container')

const nameInput = document.getElementById('name-input')

const messageForm = document.getElementById('message-form')

const messageInput = document.getElementById('message-input')

const messageTone = new Audio('/Notification.mp3')

//number of connected clients

socket.on('clients-total', (data) => {

clientsTotal.innerText = `Total Clients: ${data}`

})

function addMessageToUI(isOwnMessage, data) {

clearFeedback()

const element = `

<li class="${isOwnMessage ? 'message-right' : 'message-left'}">

<p class="message">

${data.message}

<span>${data.name} ● ${moment(data.dateTime).fromNow()}</span>

</p>

</li>

`

messageContainer.innerHTML += element

scrollToBottom()

}

function scrollToBottom() {

messageContainer.scrollTo(0, messageContainer.scrollHeight)

}

messageForm.addEventListener('submit', (e) => {

e.preventDefault()

sendMessage()

})

function sendMessage() {

if (messageInput.value === '') return

// console.log(messageInput.value)

const data = {

name: nameInput.value,

message: messageInput.value,

dateTime: new Date(),

}

socket.emit('message', data)

addMessageToUI(true, data)

messageInput.value = ''

}

socket.on('chat-message', (data) => {

console.log(data)

messageTone.play()

addMessageToUI(false, data)

})

//Handle feedback from here

messageInput.addEventListener('focus', (e) => {

socket.emit('feedback', {

feedback: `${nameInput.value} is typing a message`,

})

scrollToBottom()

})

messageInput.addEventListener('keypress', (e) => {

socket.emit('feedback', {

feedback: `${nameInput.value} is typing a message`,

})

scrollToBottom()

})

messageInput.addEventListener('blur', (e) => {

socket.emit('feedback', {

feedback: '',

})

scrollToBottom()

})

socket.on('feedback', (data) => {

clearFeedback()

const element = `

<li class="message-feedback">

<p class="feedback" id="feedback">${data.feedback}</p>

</li>

`

messageContainer.innerHTML += element

scrollToBottom()

})

function clearFeedback() {

document.querySelectorAll('li.message-feedback').forEach((element) => {

element.parentNode.removeChild(element)

})

}

\*\*\*style.css\*\*

@import url('https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;700&display=swap');

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

scroll-behavior: smooth;

}

body {

font-family: Verdana;

display: grid;

place-items: center;

background: radial-gradient(red,white);

}

.title {

margin: 20px 0px;

}

.main {

border: 8px double red;

border-radius: 24px;

overflow: hidden;

width: 80%;

}

.name {

display: flex;

font-size: 32px;

padding: 8px 16px;

color: blue;

background:lightgreen

}

.name > span {

color: navy;

}

.name-input {

font-size: 24px;

font-weight: bold;

color: blueviolet;

flex-grow: 1;

border: none;

margin: 0px 12px;

outline: none;

background:none;

}

.message-container {

display: flex;

flex-direction: column;

background:linear-gradient(to top, white 80%, pink);

width: 100%;

height: 400px;

overflow-y: scroll;

overflow-x: hidden;

}

.message-left,

.message-right {

list-style: none;

padding: 8px 12px;

margin: 12px;

max-width: 75%;

font-size: 18px;

word-wrap: break-word;

}

.message-left {

border-radius: 20px 20px 20px 0px;

align-self: flex-start;

background-color: lightpink;

box-shadow: -2px 2px 4px darkblue;

color: black;

}

.message-right {

border-radius: 20px 20px 0px 20px;

align-self: flex-end;

background-color:black;

box-shadow: 2px 2px 4px darkblue;

color:lightpink;

}

.message-left > p > span,

.message-right > p > span {

display: block;

font-style: italic;

font-size: 12px;

margin-top: 4px;

}

.feedback {

font-style: italic;

font-size: 14px;

text-align: center;

background:linear-gradient(to left, white, skyblue, white);

}

.message-form {

display: flex;

justify-content: space-between;

width: 100%;

}

.message-input {

flex-grow: 1;

height: 48px;

font-size: 16px;

border: none;

outline: none;

padding: 0 12px;

background-color: rgba(215,215,215,1);

}

.send-button {

height: 48px;

font-size: 16px;

width: 20%;

border: none;

padding: 0px 20px;

outline: none;

background-color: rgba(215,215,215,1);

cursor: pointer;

}

.send-button:hover{

background: radial-gradient(white,lightgray);

}

.v-divider {

height: 48px;

width: 2px;

background-color: lightsalmon;

}

.clients-total {

margin: 20px 0;

color: #7e7e7e;

}

================================================================

API Gateways

================================================================

**API Gateways**

* in Node.js, an API Gateway is a server that acts as an entry point for all client requests to your microservices or backend services.
* It's a critical component in microservices architectures, where you have multiple small services working together to fulfill the requirements of an application.
* API Gateways in Node.js provide a centralized and standardized way to manage communication between clients and your backend services, improving scalability, security, and maintainability of your application architecture.

<>

apiGateways.js

\*\*\* apiGateways.js\*\*\*

const express = require('express');

const router = express.Router();

// Middleware for authentication

const authenticate = (req, res, next) => {

console.log("Inside Authentication");

next();

};

// Middleware for request logging

const logRequest = (req, res, next) => {

console.log(`Received ${req.method} request to ${req.url}`);

next();

};

router.use(authenticate);

router.use(logRequest);

module.exports = router;

\*\*\*server.js\*\*\*

//import apiGateway

const apiGateway = require('./apiGateway');

//use apiGateway

app.use(apiGateway)

//import routes

. . .

================================================================

================================================================