CHAT APPLICATION

Version 1.1

Created on 23-05-2018

Contents

[INTRODUCTION 4](#_Toc517096384)

[Purpose 4](#_Toc517096385)

[Intended audience and reading 4](#_Toc517096386)

[Project Scope and features 4](#_Toc517096387)

[OVERALL DESCRIPTION 4](#_Toc517096388)

[Product Perspective 4](#_Toc517096389)

[Product features 4](#_Toc517096390)

[Assumptions and dependencies 4](#_Toc517096391)

[About node js 5](#_Toc517096392)

[Advantages of node js 5](#_Toc517096393)

[Where to use node js 6](#_Toc517096394)

[Disadvantage: 6](#_Toc517096395)

[Application Development History 6](#_Toc517096396)

[FUNCTIONAL REQUIREMENTS 8](#_Toc517096397)

[User registration 8](#_Toc517096398)

[Chat types 8](#_Toc517096399)

[Sending messages 8](#_Toc517096400)

[User details 8](#_Toc517096401)

[NON-FUNCTIONAL REQUIREMENTS 8](#_Toc517096402)

[Performance requirements 8](#_Toc517096403)

[Security requirements 8](#_Toc517096404)

[Failures 9](#_Toc517096405)

[Accuracy 9](#_Toc517096406)

[EXTERNAL INTERFACE 9](#_Toc517096407)

[IMPLEMENTATION DETAILS 9](#_Toc517096408)

[TESTING AND DEBUGGING 9](#_Toc517096409)

[KNOWLEDGE AcQUIRED THROUGH CHAT APPLICATION: 9](#_Toc517096410)

[Sample Screens 20](#_Toc517096411)

[Screen 1 20](#_Toc517096412)

[Screen 2 20](#_Toc517096413)

# INTRODUCTION

## Purpose

The project is to create a chat application which is used for different users to interact with each other. The chat application is with a server. The users can message each other immediately using this app. This is the version 1.0. The scope of the project extends to everyone. No technical knowledge is required.

## Intended audience and reading

This document is intended for software developers, marketing staff, users and testers.

## Project Scope and features

-The chat app enables communication between different people all over the world

-The objective of the chat app is to enable communication which is authenticated, secure and with timeliness.

# OVERALL DESCRIPTION

## Product Perspective

-The product is not an original version and previous similar works exist which may be found on playstore.

-Beta testing will be performed.

## Product features

-The chat app should allow unicast or multicast of messages.

## Assumptions and dependencies

No assumptions are made. This app is suitable even to beginners with no technical knowledge

## About node js

Node.js® is a JavaScript runtime environment built on Chrome’s V8 JavaScript engine.

It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js’ package ecosystem, npm, is the largest ecosystem of open source libraries in the world.  Node.js is a JavaScript runtime environment that achieves low latency and high throughput by taking a “non-blocking” approach to serving requests. In other words, Node.js wastes no time or resources on waiting for I/O requests to return.

A Node module is a reusable block of code whose existence does not accidentally impact other code.

### Advantages of node js

* Open source server environment, Platform independent
* Asynchronous and Event Driven − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* Very Fast − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* Single Threaded but Highly Scalable − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.

### Where to use node js

Following are the areas where Node.js is used.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

### Disadvantage:

It is not advisable to use Node.js for CPU intensive applications.

Not well suited for multi threaded tasks.

### Application Development History

**What was used a few years back? Java and .NET. But node js is preferable for the following reasons:**

Large web driven players in the market are moving to Node.js for these compelling reasons –

1. Node.js does not use the concept of multithreading like JAVA and this feature allows hassle free programming properties. But it does use the concept of asynchronous execution of Input output-based events through a thread pool.
2. Node.js specializes in execution and topping well for low-CPU, highly I/O-bound operations. Just starting to work on Node.js will allow a programmer to analyse how to exploit it for maximum performance.
3. Node.js has proved to work over cloud environments and client virtual machines though dedicated software. People with experience in JavaScript can easily pick up Node.js to produce very specific results. Core Node.js scripts can perform the business logic straight at the server side.
4. Though .NET is powerful, Node.js is much easier to use and more accessible to developers. Node.js focusses in executing and scaling better in low-CPU based systems and highly I/O-bound operations.Node.js can work efficiently on heavily I/O-bound operations with low CPU usage.
5. Comparing on the factors of language and package, Node.js uses less coding to perform tasks as compared to Java and .Net. It also wins on open source libraries though they are most driven on communities around.
6. Node.js has the big advantage of code re-use. This feature makes it a big hit among developers. Though it is limited by the server resources, its contention feature used for app development works regardless of thread delegation. This is resource contention is a big hit among developers. Node.js uses an event-based paradigm, and .NET does also when implemented asynchronously.
7. Node.js projects are proven to compile within few minutes. Most of the Node.js test cases are easily available thereby giving code re-use possibility and feature sharing for various similar subjects and applications. Node.js is a clear winner when controlling stack overflow in server side scripting.

Comparing Node.js with JAVA or .Net completely relies on a customer’s need and the software design to be built. Node.js performs better and much faster compared to Java or .Net.

# FUNCTIONAL REQUIREMENTS

## User registration

The end-user should register to the app using a user name. He will remain signed in until user logs out explicitly or the network has problems.

## Chat types

The chat rooms are global. Multiple users(maximum of 50) can participate in a global chat and can leave at any time. The user online list is displayed. The admin can create or delete global chats and set background for global chats.

## Sending messages

Users can send messages to anyone through global chats. Only texts can be sent. User gets the messages displayed on screen when it arrives.

## User details

Each user has a unique name which cannot be changed.

# NON-FUNCTIONAL REQUIREMENTS

The non-functional requirements include

## Performance requirements

The application should have low latency and high throughput regardless of the increase in the number of users.

## Security requirements

The chat room should only be available to those who establish connection with the server.

## Failures

In case of system failure and crashes, the back up of chat must be provided from the database.

## Accuracy

The message should have timely delivery and accuracy.

# EXTERNAL INTERFACE

The interface used is user interface (Chrome browser) or GUI.

# IMPLEMENTATION DETAILS

Coding is done, and programming language is used. It is coded and compiled.

The programming languages used are HTML, CSS and JS using Node Js.

Sublime Text 3 editor and database is also used.

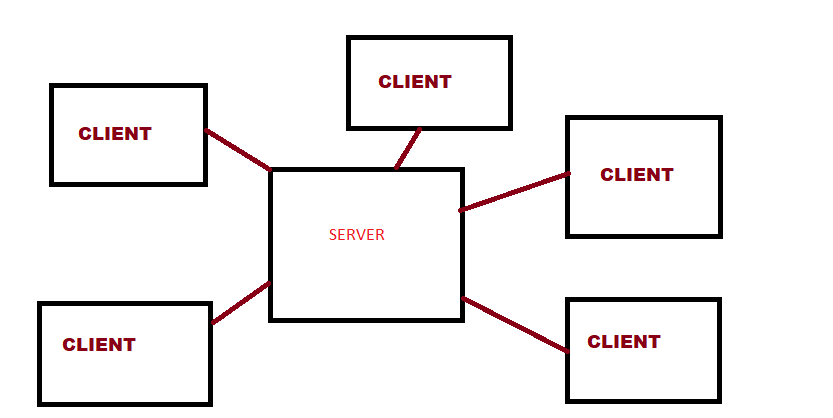
# TESTING AND DEBUGGING

Testing can be positive or negative. Unit testing, validation, integration testing and regression testing are done with appropriate test cases followed by debugging.

# KNOWLEDGE AcQUIRED THROUGH CHAT APPLICATION:

* To make a chat app using node js
* HTML
* CSS and layouts
* Node js and npm
* How to create a server
* Establishing socket connection between different clients and server
* Express framework and Web sockets in node js
* Methods in web sockets and connection close
* How to create an online-user list
* To send message from server to clients and vice versa
* Broadcast messages
* DOM manipulation in javascript
* Event Handling

Several clients can interact with a single server. Clients can fetch data from or send data to server and cannot interact directly with each other.



Initially, a HTML page called index.html is made in the public folder which displays when you visit port 5000. The styles are rendered by styles.css file.

These two files are static files and are in the public directory.

Npm init is entered in the cmd and a package.json file appears. The data is filled and following dependencies are installed:

npm install express -save

npm install socket.io -save

npm install nodemon

npm install body-parser -save

Save allows the dependencies to be saved automatically in package,json.

The server runs on app.js file.

* Initially, the express framework is required and stored in variable express and socket.io is stored in socket.
* The port is defined and express is set up by calling function to variable app.
* The var app now listens to the port by app.listen() and a call-back function is fired.
* Static files are rendered(html and css files)
* The socket connection is made to the clients. This is handled by passing parameter server, which listens to port, to socket() and storing in the var io.
* When any client enters user name, it listens to the message ‘name’ and emits all the names received so far in an array to all the clients. This allows clients to know who are online and who are not.
* When any client sends message to server, it listens for ‘msg’ and emits the message received to all the clients (for global chat);
* Server listens to ‘typing’ and emits that the particular server (distinguished by the user name), is typing a message to all other clients except the sending client which is **broadcasting**.
* Server listens to ‘disconnect’ to know if a client quit the tcp connection and informs so to all other clients by emitting ‘close’. The clients can’t update their online lists accordingly and display the messages only amongst each other.

//SERVER

var express = require('express');

var socket = require('socket.io');

var port = 5000;

//set up express

var app = express();

//listen to port

var server = app.listen(port, function() {

console.log('Listening to port '+port);

});

//static files

app.use(express.static('public'));

//socket connection

var io = socket(server);

var ar=[''];

var i=0;

let username =[''];

io.on('connection', function(socket) {

console.log('Connection made with ' + socket.id);

socket.on('name', function(data) {

ar.push(data);

console.log('Recieved by server ' + ar);

username[socket.id] = data;

io.sockets.emit('userName',ar);

//console.log('key value pair '+username[socket.id]);

i++;

});

socket.on('msg', function(ob) {

console.log('Recievd the object '+ ob.user +' '+ob.text);

io.sockets.emit('texts',ob);

});

socket.on('typing', function(user) {

socket.broadcast.emit('typing', user);

});

socket.on('disconnect', function() {

console.log(' disconnected by ' + username[socket.id]);

console.log(ar);

var diconn\_userIndex = ar.indexOf(username[socket.id]);

ar.splice(diconn\_userIndex,1);

socket.broadcast.emit('close', {u: username[socket.id], ar: ar});

console.log(ar);

});

});

The html page rendered is chat.html. Here, users can type and send the message.

The client side js program is as follows:

* The client also establishes connection using web sockets with server.
* The uname is defined with the user name entered in index.html.
* The client emits ‘name’ to server.
* The dom is queried by using ids from the html document.
* The client listens to ‘userName’ and receives the array. By using indexOf() method, if value is negative, the user is not in array. By using this logic, the online list is generated.
* On listening to ‘close’ message, the clients are informed that a particular user has closed the connection with the server. That particular username is removed from the online list by using splice method.
* Whatever is typed in the text box is added to the output window and the message is sent to server. Server broadcasts this to all other clients and the clients output the message they receive to the window. This allows all clients to view the message which has been typed.
* The ‘typing’ allows the client to sent the username typing the current message. This is broadcasted and all other clients display that the user is typing the message.

//CLIENT

//make connection with server

var socket = io.connect('http://localhost:5000');

var uname = localStorage.getItem('userName');

//query dom

document.querySelector('#u-name').textContent = uname;

var onlineList = document.querySelector('#online ul');

let output = document.getElementById('out');

let text = document.getElementById('message');

let sendbtn = document.querySelector('#send');

var feedback = document.getElementById('feedback');

socket.emit('name', uname);

var nameList=[''];

var x=[''];

var nameL=[''];

nameList[0]=uname;

nameL[0]=uname;

let lis = new Set();

//add to online list

socket.on('userName', function(ar) {

ar.forEach(function(data) {

console.log(data);

if(nameList.indexOf(data) == -1) { //if data is not in nameList

onlineList.innerHTML +='<li>' + data + '</li>';

nameList.push(data);

}

});

});

//remove people from online list

socket.on('close', function(ob) {

onlineList.innerHTML ="";

nameL.push(ob.u);

var x=[''];

console.log(ob.u + ' closed connection');

for(var i=0; i<nameL.length; i++) {

x.push(nameL[i]);

}

Array.from(ob.ar).forEach(function(data) {

//if data is not in nameList

if(x.indexOf(data) == -1) {

onlineList.innerHTML +='<li>' + data + '</li>';

x.push(data);

}

});

});

//add what you type to output

//when you select the send button

sendbtn.addEventListener('click', function() {

msg();

});

//when you hit enter key

text.addEventListener('keyup',function(e) {

if((e.keyCode || e.which)=='13') {

msg();

}

});

var msg = function() {

socket.emit('msg', {text: text.value, user : uname} );

text.value='';

};

socket.on('texts', function(ob) {

feedback.innerHTML='';

output.innerHTML += '<p><strong>'+ ob.user + ': </strong>' +ob.text + '</p>';

});

//to display typing

text.addEventListener('keypress', function(e){

socket.emit('typing', uname);

console.log('from client, user name is ' + uname);

});

socket.on('typing', function(usr) {

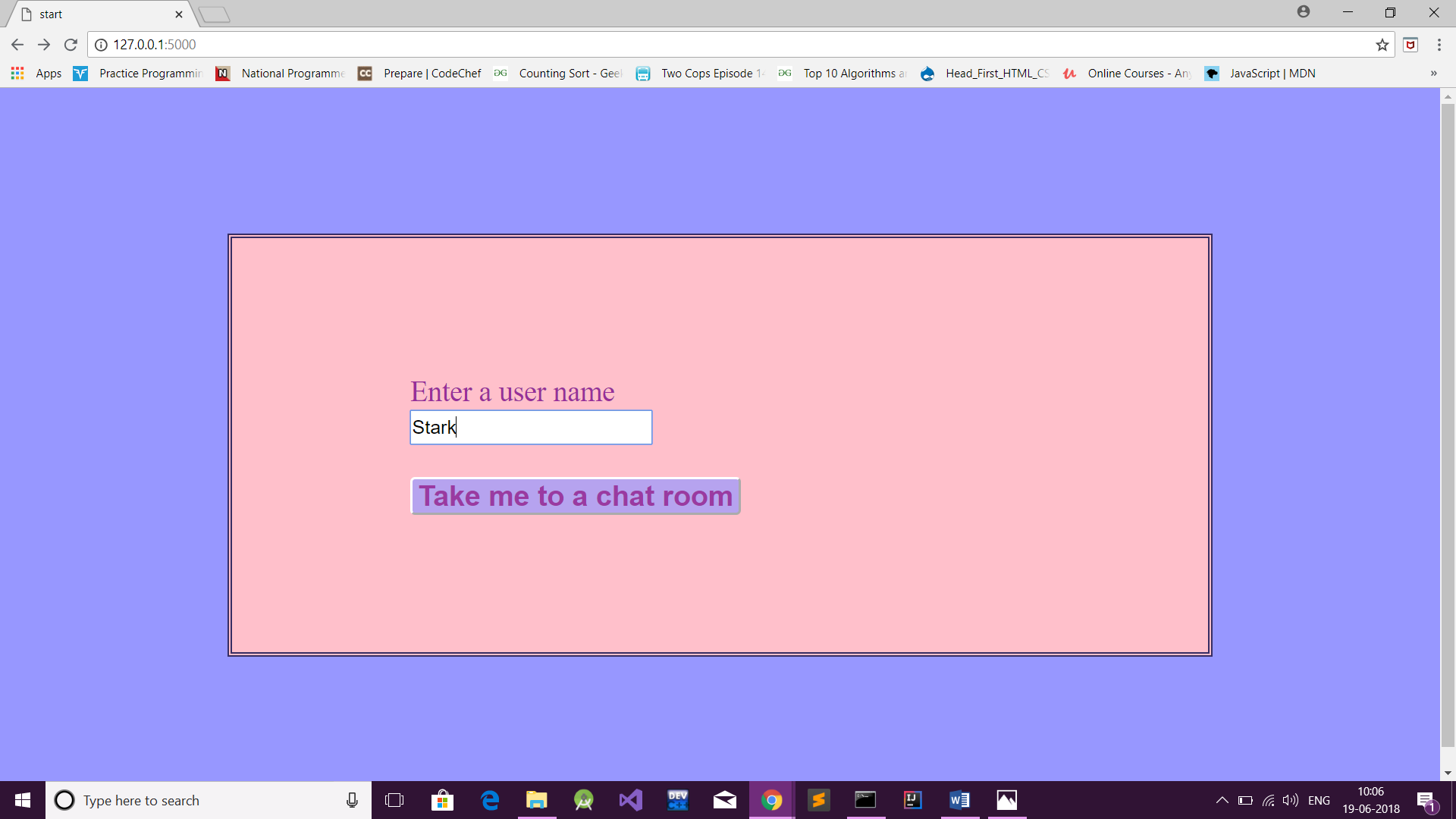
feedback.innerHTML = '<p><em>' + usr + ' is typing.. </em></p>';

});

Thus a global chat app is implemented using web sockets and express framework.

# Sample Screens

## Screen 1



## Screen 2

