LABWORK
COURSE: DISTRIBUTED SYSTEMS
CHAPTER 3: Processes and Threads

1. Develop a Chat room in using socket.io

1.1. Contents

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C.
Socket.IO is a JavaScript library for realtime web applications. It enables realtime, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API. Like Node.js, it is event-driven. Socket.IO primarily uses the WebSocket protocol with polling as a fallback option, while providing the same interface. Although it can be used as simply a wrapper for WebSocket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O.
In this labwork you will learn to use library socket.io (node.js language) to develop a web chat room in basing on protocol WebSocket. In fact, it's a real-time system of sending and receiving data. You cannot do it in using traditional way because in the classic web, a client requests to a server and a server responds sending it back the data. This system is impossible for a chat app. In WebSockets, the server can send data to the client, but the client can too! A WebSocket is a kind of communication pipe opened in two directions. You have to use WebSocket with the library socket.io to realize it.

1.2. Requirements

1.2.1. Theory

- WebSockets and library socket.io
- node.js

1.2.2. Hardwares

- Laptop/PC on Windows

1.2.3. Softwares

- node.js

1.3. PRACTICAL STEPS

First, you have to download and install node.js:
https://nodejs.org/en/download/

Node.js is a Javascript back-end technology executed by the server as PHP, Ruby or Python. JavaScript uses events. Node.js keeps this particularity so it is easy to make asynchronous code. Node.js comes with its own package manager: npm. It becomes easy to install, update, and delete packages. In this labwork, you are going to use express.js. It's a micro web framework based on node.js.

Now, create a folder called ChatRoomApp, go into it and initialize the development environment:

```
>mkdir ChatRoomApp
>cd ChatRoomApp
>npm init
```

You can skip all the questions by pressing enter until the end.

Question 1: Which file just appears in the folder ChatRoomApp? What is it used for?

Now you have to install some packages needed to develop your chat room.

```
>npm install --save express
>npm install --save nodemon
>npm install --save ejs
>npm install --save socket.io
```

You have just install 4 packages:

- *express*: the micro web application framework for node.js
- *nodemon*: a package that will detect any changes and restart your server. You will use it instead of the classic node command.
- *ejs*: a template engine to simplify the production of HTML
- *socket.io*: the famous package to manage WebSockets

In your package.json, you change your scripts key as below:

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app"
  }
```

You have to develop two parts: the client part and the server part.
You create a file called *app.js* that will launch our server and all the packages.
Create a new file app.js and insert the text below:

```
const express = require('express')
const app = express()
```

```
//set the template engine ejs
app.set('view engine', 'ejs')

//middlewares
app.use(express.static('public'))


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

//Listen on port 3000
server = app.listen(3000)
```

Now, you can launch your app with the command:

```
>npm run start
```

Question 2: Try to open your web browser and go to *http://localhost:3000*, which message do you see?

Insert these lines to the file app.js:

```
//socket.io instantiation
const io = require("socket.io")(server)


//listen on every connection
io.on('connection', (socket) => {
         console.log('New user connected')
})
```

Here, the *io* object will give us access to the *socket.io* library. *io* object is now listening to each connection to your app. Each time a new user is connecting, it will print out "New user connected" in your console.

Question 3: You try to reload your web browser. Do you see something new in your console window? If nothing happened, why?

For the moment socket.io is only installed on the server part. Next, we will do the same work on the client side.

You just have to change a line in your *app.js*. In fact, you don't want to display a "Hello world" message anymore, but a really window with a chat box, inputs to write username/message and a send button. To do that you have to render an html file (which will be in your case an *ejs* file) when accessing the "/" root.

You change the routes part of the file *app.js* as below:

```
//routes
```

```
app.get('/', (req, res) => {
        res.render('index')
})
```

In the your folder ChatRoomApp, create 2 new subfolders called *public* and *views*.

```
>mkdir public
>mkdir views
```

In the folder *views*, create a file named *index.ejs* with the content as below:
(or you can download it with this link:
https://github.com/anhth318/ChatRoomApp/blob/master/views/index.ejs)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" const="text/html;charset=UTF-8"
/>
    <link href="http://fonts.googleapis.com/css?family=Comfortaa"
rel="stylesheet" type="text/css">
    <link rel="stylesheet" type="text/css" href="style.css" >
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.4/socket.io
.js"></script>
    <title>Distributed Systems Course</title>
  </head>

  <body>
    <header>
      <h1>Chat room for Distributed Systems class</h1>
    </header>

    <section>
      <div id="change_username">
                   <input id="username" type="text" />
                   <button id="send_username" type="button">Change
username</button>
      </div>
    </section>

    <section id="chatroom">
      <section id="feedback"></section>
    </section>



    <section id="input_zone">
      <input id="message" class="vertical-align" type="text" />
      <button id="send_message" class="vertical-align"
type="button">Send</button>
    </section>

    <script src="http://code.jquery.com/jquery-
latest.min.js"></script>
    <script src="chat.js"></script>
  </body>
</html>
```

In the folder *public*, create a file *chat.js* with the content as below:

```
$(function(){
     //make connection
     var socket = io.connect('http://localhost:3000')
});
```

(make attention that if you are working on two separated PCs, you can replace the *localhost* with the IP of the server).

Download the file *style.css* and place it in the folder *public*:
https://github.com/anhth318/ChatRoomApp/blob/master/public/style.css

Question 4: Refresh the page localhost:3000, which message do you see in the console window?

In the client side, you want now to control some different actions like: emit message, listen on new message, and emit a username. Change the source code of *chat.js* as follows:

```
$(function(){
   //make connection
   var socket = io.connect('http://localhost:3000')

   //buttons and inputs
   var message = $("#message")
   var username = $("#username")
   var send_message = $("#send_message")
   var send_username = $("#send_username")
   var chatroom = $("#chatroom")

   //Emit message
   send_message.click(function(){
       socket.emit('new_message', {message : message.val()})
   })

   //Listen on new_message
   socket.on("new_message", (data) => {
       message.val('');
       chatroom.append("<p class='message'>" + data.username + ": " +
data.message + "</p>")
   })

   //Emit a username
   send_username.click(function(){
          socket.emit('change_username', {username : username.val()})
   })
});
```

In the file app.js, edit the code of the part of *listen on every connection* as below:

```
//listen on every connection
io.on('connection', (socket) => {
    console.log('New user connected')

    //default username
    socket.username = "Anonymous"

    //listen on change_username
    socket.on('change_username', (data) => {
          socket.username = data.username
    })

    //listen on new_message
    socket.on('new_message', (data) => {
        //broadcast the new message
        io.sockets.emit('new_message', {message : data.message, 
username : socket.username});
    })
})
```

For the *new_message* event, you can see that we call the sockets property of *io*. It represents all the sockets connected. So this line will actually send a message to all the sockets. We want that to show a message sent by a user to all (and itself included).

Now, it’s time to test your app. Open the url http://localhost:3000/ in several tabs of your web browser, try to chat in the room.

If you want to make your chat room more realistic, you can make it appear “*UserA is typing…*” message when someone is typing.

Open the file *app.js*, insert the code below in the io.on(…) function:

```
//listen on typing
socket.on('typing', (data) => {
         socket.broadcast.emit('typing', {username : 
socket.username})
})
```

Open the file chat.js, add some code as below:

In the part of “//buttons and inputs”, add the line:

```
var feedback = $("#feedback")
```

In the part of “//Listen on new_message”, edit the code as below:

```
//Listen on new_message
socket.on("new_message", (data) => {
    feedback.html('');
    message.val('');
    chatroom.append("<p class='message'>" + data.username + ": " + 
data.message + "</p>")
})
```

Insert two parts as below:

```
//Emit typing
message.bind("keypress", () => {
    socket.emit('typing')
})

//Listen on typing
    socket.on('typing', (data) => {
```

```
    feedback.html("<p><i>" + data.username + " is typing a
message..." + "</i></p>")
})
```

It's done! Now, you can refresh your windows to test your chat room.

Question 5: Now you try to type something in one tab. In the same time, look at other chat window tab of other user, what do you see?