



Modul 09

Socket Programming

TUJUAN PEMBELAJARAN

1. Mahasiswa mampu mengimplementasikan komunikasi real-time menggunakan pendekatan Socket Programming dengan WebSocket.

HARDWARE & SOFTWARE

1. PC (*Personal Computer*) dengan akses Internet
2. JavaScript
3. Visual Studio Code atau IDE lainnya yang mendukung JavaScript
4. Node.js
5. Socket.io

URAIAN MATERI

A. Socket Programming

Socket programming adalah paradigma pemrograman yang memungkinkan komunikasi antarproses melalui jaringan komputer menggunakan soket (socket). Soket merupakan endpoint dari saluran komunikasi yang memungkinkan dua proses yang berjalan pada mesin yang berbeda untuk saling bertukar data. Paradigma ini sangat penting dalam pengembangan aplikasi jaringan, di mana berbagai entitas perangkat lunak perlu berkomunikasi satu sama lain melalui jaringan.

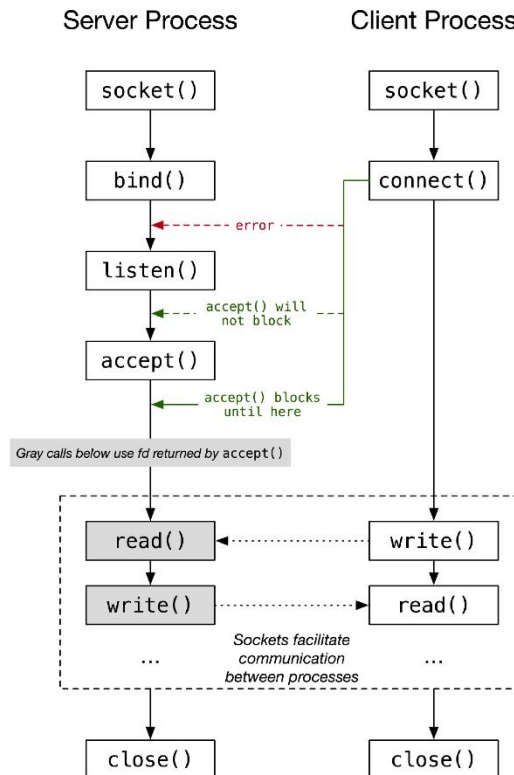
Socket programming memungkinkan aplikasi untuk mengirim dan menerima data melalui jaringan, baik melalui protokol TCP (Transmission Control Protocol) yang dapat diandalkan dan berorientasi koneksi, maupun protokol UDP (User Datagram Protocol) yang lebih sederhana dan tidak berorientasi koneksi. Melalui socket, aplikasi dapat mengakses layanan jaringan, berbagi informasi, dan melakukan komunikasi terdistribusi.

Pengembangan aplikasi dengan menggunakan socket programming melibatkan implementasi kelas-kelas atau fungsi-fungsi yang memungkinkan pembuatan, pengelolaan, dan penggunaan soket. Proses ini melibatkan pembuatan soket, pengikatan soket ke alamat dan port tertentu, mendengarkan koneksi masuk (untuk server), dan menginisiasi koneksi (untuk

klien). Selain itu, socket programming juga melibatkan pengiriman dan penerimaan data melalui soket, serta penanganan pesan atau paket data yang diterima.

State Diagram Untuk Model Server dan Client

Perhatikan diagram berikut untuk memahami bagaimana socket bekerja



Berikut adalah penjelasan gambar diatas

1. Node dibagi menjadi dua jenis, node server dan node klien.
2. Node klien mengirimkan sinyal koneksi dan node server menerima sinyal koneksi yang dikirim oleh node klien.
3. Koneksi antara node server dan klien dibangun menggunakan soket di lapisan transport internet.
4. Setelah koneksi terbentuk, node klien dan server dapat berbagi informasi di antara mereka menggunakan perintah baca dan tulis.
5. Setelah pertukaran informasi selesai, node menutup koneksi.

B. WebSocket

WebSocket adalah protokol komunikasi komputer yang menyediakan saluran komunikasi dua arah secara simultan melalui satu koneksi Transmission Control Protocol (TCP). Protokol WebSocket telah distandardisasi oleh IETF (Internet Engineering Task Force) sebagai RFC 6455 pada tahun 2011. Informasi lengkap tentang standar ini dapat dibaca melalui link berikut

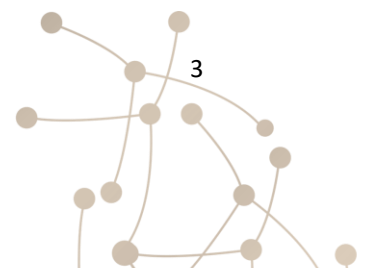
ini <https://datatracker.ietf.org/doc/html/rfc6455>. Spesifikasi saat ini yang memungkinkan aplikasi web menggunakan protokol ini dikenal sebagai *WebSockets*. Hal ini merupakan standar yang terus berkembang yang dikelola oleh WHATWG (Web Hypertext Application Technology Working Group) dan menjadi penerus dari API *WebSocket API* dari W3C (World Wide Web Consortium).

WebSocket berbeda dari Hypertext Transfer Protocol (HTTP) yang digunakan untuk melayani sebagian besar halaman web. Kedua protokol ini berada pada lapisan 7 dalam model OSI dan bergantung pada TCP pada lapisan 4. Meskipun berbeda, RFC 6455 menyatakan bahwa WebSocket dirancang untuk berfungsi melalui port HTTP 443 dan 80 serta mendukung proksi dan perantara HTTP, sehingga membuatnya kompatibel dengan HTTP. Untuk mencapai kompatibilitas, *WebSocket handshake* menggunakan *HTTP upgrade header* untuk beralih dari protokol HTTP ke protokol WebSocket.

WebSocket memungkinkan komunikasi dua arah secara langsung dan real-time antara klien dan server. Hal ini berbeda dengan protokol HTTP tradisional yang bersifat stateless dan membutuhkan permintaan dari klien untuk setiap respons dari server. Setelah koneksi WebSocket dibuat, koneksi tersebut tetap terbuka dan dapat digunakan untuk pertukaran pesan secara terus-menerus. Hal ini mengurangi overhead yang terkait dengan membuka dan menutup koneksi yang sering terjadi dalam protokol HTTP. WebSocket juga memiliki overhead yang lebih rendah dibandingkan dengan HTTP karena tidak perlu membuka koneksi baru untuk setiap pertukaran pesan. Overhead adalah istilah yang digunakan untuk merujuk pada biaya tambahan atau beban ekstra yang timbul ketika menjalankan suatu tindakan atau proses. Dalam konteks teknologi informasi dan komunikasi, overhead dapat merujuk pada penggunaan sumber daya tambahan yang tidak langsung terlibat dalam tugas utama.

Protokol WebSocket dirancang untuk memiliki overhead minimal sehingga membuatnya cocok untuk situasi di mana penghematan bandwidth dan respons waktu yang cepat sangat penting. Hal ini membuatnya sangat efisien untuk aplikasi yang membutuhkan komunikasi real-time, seperti aplikasi obrolan (chat) atau pembaruan data langsung. Selain itu, WebSocket dapat mendukung koneksi cross-domain (dengan menggunakan CORS), yang memungkinkan aplikasi berbasis web untuk berkomunikasi dengan server di domain yang berbeda.

Mozilla telah mendokumentasikan WebSockets melalui website resmi Mozilla Developer Networks (MDN) di link berikut ini https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. Berdasarkan penjelasan Mozilla, API WebSocket adalah teknologi canggih yang memungkinkan pembukaan sesi komunikasi interaktif dua arah antara browser pengguna dan server. Dengan API ini, Anda dapat mengirim pesan ke server dan menerima respons yang dipicu oleh peristiwa tanpa perlu melakukan pengecekan terus-menerus ke server untuk mendapatkan balasan.



C. Socket.io

Socket.io adalah salah satu tools yang mendukung socket programming menggunakan WebSockets dan didesain khusus untuk Node.js. Anda dapat menelusuri berbagai tools lain yang digunakan pada platform atau Bahasa pemrograman lainnya pada dokumentasi WebSocket Mozilla. Socket.IO adalah library yang memungkinkan komunikasi dua arah (**bidirectional**) berbasis peristiwa (**event-based**) dengan latensi rendah (**low-latency**) antara klien dan server. Saluran dua arah (bidirectional channel) antara server Socket.IO (Node.js) dan klien Socket.IO (browser, Node.js, atau bahasa pemrograman lainnya) dibangun dengan koneksi WebSocket kapanpun dibutuhkan, dan akan menggunakan long-polling HTTP sebagai solusi cadangan (fallback).

Basis kode Socket.IO terbagi menjadi dua lapisan yang berbeda:

1. Pipa dasar (*low-level plumbing*) yang disebut Engine.IO, adalah mesin di dalam Socket.IO.
2. Antarmuka API tingkat tinggi (high-level API) yaitu Socket.IO itu sendiri.

Engine.IO bertanggung jawab untuk membentuk koneksi tingkat rendah antara server dan klien. Hal ini menangani berbagai perpindahan dan mekanisme upgrade, deteksi pemutusan koneksi. Versi rinci dari protokol Engine.IO dapat ditemukan melalui link berikut ini <https://socket.io/docs/v4/engine-io-protocol/>. Adapun kode sumber implementasi referensi (ditulis dalam TypeScript) dapat ditemukan di sini: :

- server: <https://github.com/socketio/engine.io>
- client: <https://github.com/socketio/engine.io-client>
- parser: <https://github.com/socketio/engine.io-parser>

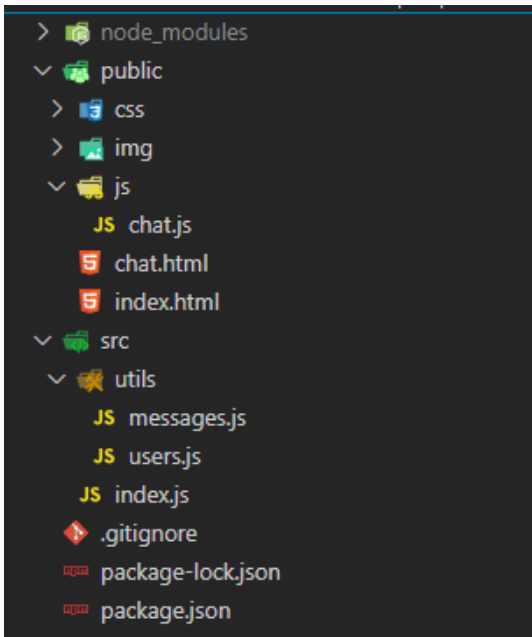
Sementara itu, Socket.IO menyediakan beberapa fitur tambahan di atas koneksi Engine.IO seperti rekoneksi otomatis, penyimpanan paket sementara, pengakuan (acknowledgments), penyiaran ke semua klien atau ke sebagian klien (yang disebut "Room"), multiplexing (yang disebut "Namespace"). Adapun versi rinci dari protokol Socket.IO dapat dibaca melalui link berikut ini <https://socket.io/docs/v4/socket-io-protocol/>

Kode sumber implementasi referensi (ditulis dalam TypeScript) dapat ditemukan di sini:

- server: <https://github.com/socketio/socket.io>
- klien: <https://github.com/socketio/socket.io-client>
- parser: <https://github.com/socketio/socket.io-parser>

LATIHAN

1. Buatlah sebuah folder baru dengan nama '**ruangobrol**'. Berikut adalah **struktur file** dan **folder** dalam folder tersebut



2. Pada file **index.js** di folder **src**. Masukkanlah source code berikut ini

```
const path = require('path')
const http = require('http')
const express = require('express')
const socketio = require('socket.io')
const Filter = require('bad-words')
const {generateMessage, generateLocationMessage} = require('./utils/messages')
const {tambahPengguna, hapusPengguna, ambilPengguna, ambilPenggunaDariRoom} =
require('./utils/users')

const app = express()
const server = http.createServer(app)
const io = socketio(server)

const port = process.env.PORT || 3000;
const publicDirectoryPath = path.join(__dirname, '../public')

app.use(express.static(publicDirectoryPath))

io.on('connection', (socket) => {
  console.log('New WebSocket connection')

  socket.on('join', (options, callback) => {
    const {error, user} = tambahPengguna({id: socket.id, ...options})
    if(error) {
      return callback(error)
    }
    socket.join(user.room)
    socket.emit('pesan', generateMessage('Admin', 'Selamat datang!'))
    socket.broadcast.to(user.room).emit('pesan',
generateMessage('Admin', `${user.username} telah bergabung`))
    io.to(user.room).emit('roomData', {
```

```
        room: user.room,  
        users: ambilPeggunaDariRoom(user.room)  
    })  
  
    callback()  
})  
  
socket.on(' kirimPesan', (pesan, callback) => {  
    const user = ambilPegguna(socket.id)  
    const filter = new Filter()  
  
    if(filter.isProfane(pesan)){  
        return callback('Pesan tidak boleh mengandung kata kasar')  
    }  
    io.to(user.room).emit(' pesan', generateMessage(user.username, pesan))  
    callback()  
})  
socket.on(' kirimLokasi', (coords, callback) => {  
    const user = ambilPegguna(socket.id)  
    io.to(user.room).emit(' locationMessage',  
generateLocationMessage(user.username,  
`https://www.google.com/maps?q=${coords.latitude},${coords.longitude}`))  
    callback()  
})  
  
socket.on(' disconnect', () => {  
    const user = hapusPegguna(socket.id)  
    if(user){  
        io.to(user.room).emit(' pesan',  
generateMessage('Admin', `${user.username} telah keluar`))  
        io.to(user.room).emit(' roomData', {  
            room: user.room,  
            users: ambilPeggunaDariRoom(user.room)  
        })  
    }  
})  
})  
  
server.listen(port, () => {  
    console.log(`Server is running on port ${port}!`)  
})
```

3. Pada file **users.js** di folder **src/utlis**, masukanlah source code berikut ini

```
const users = []  
  
//tambahPegguna  
const tambahPegguna = ({id, username, room}) => {  
    //clean the data  
    username = username.trim().toLowerCase()  
    room = room.trim().toLowerCase()  
  
    //validate the data  
    if(!username || !room){  
        return {
```

```
        error: 'Username dan room dibutuhkan!'
      }
    }

    //check for exisiting user
    const existingUser = users.find((user)=>{
      return user.room === room && user.username === username
    })

    //validate username
    if (existingUser){
      return {
        error: 'Username sudah digunakan!'
      }
    }

    //store user
    const user = {id, username, room}
    users.push(user)
    return {user}
  }

  //hapusPengguna
  const hapusPengguna = (id) => {
    const index = users.findIndex((user) => user.id === id)
    if(index !== -1){
      return users.splice(index, 1)[0]
    }
  }

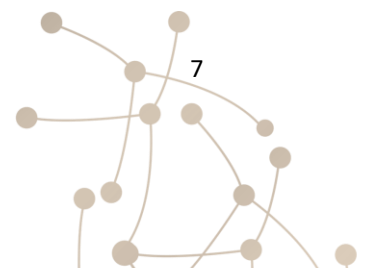
  //ambilPengguna
  const ambilPengguna = (id) => {
    return users.find((user) => user.id === id)
  }

  //ambilPenggunaDariRoom
  const ambilPenggunaDariRoom = (room) => {
    room = room.trim().toLowerCase()
    return users.filter((user) => user.room === room)
  }

  module.exports = {
    tambahPengguna,
    hapusPengguna,
    ambilPengguna,
    ambilPenggunaDariRoom
  }
}
```

4. Pada file **messages.js** di folder **src/utlis**, masukanlah source code berikut ini

```
const generateMessage = (username, text) =>{
  return {
    username,
    text,
    createdAt: new Date().getTime()
  }
}
```



```
}  
}  
  
const generateLocationMessage = (username, url) => {  
  return {  
    username,  
    url,  
    createdAt: new Date().getTime()  
  }  
}  
module.exports = {  
  generateMessage,  
  generateLocationMessage  
}
```

5. Pada file index.html di folder public, masukanlah source code berikut ini

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>kitangobrol</title>  
    <link rel="icon" href="/img/favicon.png">  
    <link rel="stylesheet" href="/css/styles.min.css">  
  </head>  
  <body>  
    <div class="centered-form">  
      <div class="centered-form__box">  
        <h1>ruangobrol</h1>  
        <form action="/chat.html">  
          <label>Display name</label>  
          <input type="text" name="username" placeholder="Display Name"  
required>  
          <label>Room</label>  
          <input type="text" name="room" placeholder="Room" required>  
          <button>Bergabung ke Chat</button>  
        </form>  
      </div>  
    </div>  
  </body>  
</html>
```

6. Pada file chat.html di folder public, masukanlah source code berikut ini

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>ruangobrol</title>  
    <link rel="icon" href="/img/favicon.png">  
    <link rel="stylesheet" href="/css/styles.min.css">  
  </head>  
  <body>  
    <div class="chat">  
      <div id="sidebar" class="chat__sidebar">  
  
      </div>  
      <div class="chat__main">
```

```
<div id="messages" class="chat__messages"></div>
<div class="compose">
  <form id="form-pesan">
    <input name="pesan" placeholder="Pesan" required
autocomplete="off">
    <button>Kirim</button>
  </form>
  <button id="kirim-lokasi">Share Lokasi</button>
</div>
</div>
</div>

<script id="message-template" type="text/html">
  <div class="message">
    <p>
      <span class="message__name">{{username}}</span>
      <span class="message__meta">{{createdAt}}</span>
    </p>
    <p>{{message}}</p>
  </div>
</script>
<script id="locationMessage-template" type="text/html">
  <div class="message">
    <p>
      <span class="message__name">{{username}}</span>
      <span class="message__meta">{{createdAt}}</span>
    </p>
    <p><a href="{{url}}" target="_blank">Lokasi saya sekarang</a></p>
  </div>
</script>

<script id="sidebar-template" type="text/html">
  <h2 class="room-title">Ruang: {{room}}</h2>
  <h3 class="list-title">Anggota Ruang</h3>
  <ul class="users">
    {{#users}}
      <li>- {{username}}</li>
    {{/users}}
  </ul>
</script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/mustache.js/3.0.1/mustache.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.22.2/moment.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/qs/6.6.0/qs.min.js"></script>
<script src="/socket.io/socket.io.js"></script>
<script src="/js/chat.js"></script>
```

```
</body>  
</html>
```

7. Pada file chat.js di folder public/js, masukanlah source code berikut ini

```
const socket = io()  
  
// Elements  
const $messageForm = document.querySelector('#form-pesan')  
const $messageFormInput = document.querySelector('input')  
const $messageFormButton = document.querySelector('button')  
const $sendLocationButton = document.querySelector('# kirim-lokasi')  
const $messages = document.querySelector('#messages')  
  
// Templates  
const messageTemplate = document.querySelector('#message-template').innerHTML  
const locationMessageTemplate = document.querySelector('#locationMessage-template').innerHTML  
const sidebarTemplate = document.querySelector('#sidebar-template').innerHTML  
  
// Options  
const {username, room} = Qs.parse(location.search, {ignoreQueryPrefix: true})  
  
const autoScroll = () => {  
  // New message element  
  const $newMessage = $messages.lastElementChild  
  // Height of the new message  
  const newMessageStyles = getComputedStyle($newMessage)  
  const newMessageMargin = parseInt(newMessageStyles.marginBottom)  
  const newMessageHeight = $newMessage.offsetHeight + newMessageMargin  
  //Visible Height  
  const visibleHeight = $messages.offsetHeight  
  //Height of messages container  
  const containerHeight = $messages.scrollHeight  
  //How far have I scrolled/  
  const scrollOffset = $messages.scrollTop + visibleHeight  
  
  if(containerHeight - newMessageHeight <= scrollOffset){  
    $messages.scrollTop = $messages.scrollHeight  
  }  
}  
  
socket.on('pesan', (message) => {  
  console.log(message)  
  const html = Mustache.render(messageTemplate, {  
    username: message.username,  
    message: message.text,  
    createdAt: moment(message.createdAt).format('H:mm')  
  })  
  $messages.insertAdjacentHTML('beforeend', html)  
  autoScroll()  
})  
  
socket.on('locationMessage', (message) => {  
  console.log(message)
```

```
const html = Mustache.render(locationMessageTemplate, {
  username: message.username,
  url: message.url,
  createdAt: moment(message.createdAt).format('H:mm')
})
$messages.insertAdjacentHTML('beforeend', html)
})

socket.on('roomData', ({room, users}) => {
  const html = Mustache.render(sidebarTemplate, {
    room,
    users
  })
  document.querySelector('#sidebar').innerHTML = html
})

$messageForm.addEventListener('submit', (e) => {
  e.preventDefault()
  //disable form button
  $messageFormButton.setAttribute('disabled', 'disabled')

  const pesan = e.target.elements.pesan.value
  socket.emit(' kirimPesan', pesan, (error) => {
    //enable form button
    $messageFormButton.removeAttribute('disabled')
    $messageFormInput.value = ''
    $messageFormInput.focus()

    if(error){
      return console.log(error)
    }
    console.log('Pesan berhasil dikirim')
  })
})

$sendLocationButton.addEventListener('click', (e) => {
  if(!navigator.geolocation){
    return alert('Browser anda tidak mendukung Geolocation')
  }

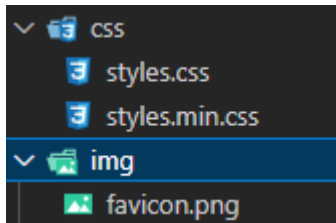
  $sendLocationButton.setAttribute('disabled', 'disabled')

  navigator.geolocation.getCurrentPosition((position) => {
    socket.emit(' kirimLokasi', {
      latitude: position.coords.latitude,
      longitude: position.coords.longitude
    }, () => {
      $sendLocationButton.removeAttribute('disabled')
      console.log('Lokasi berhasil dikirim')
    })
  })
})
})
```

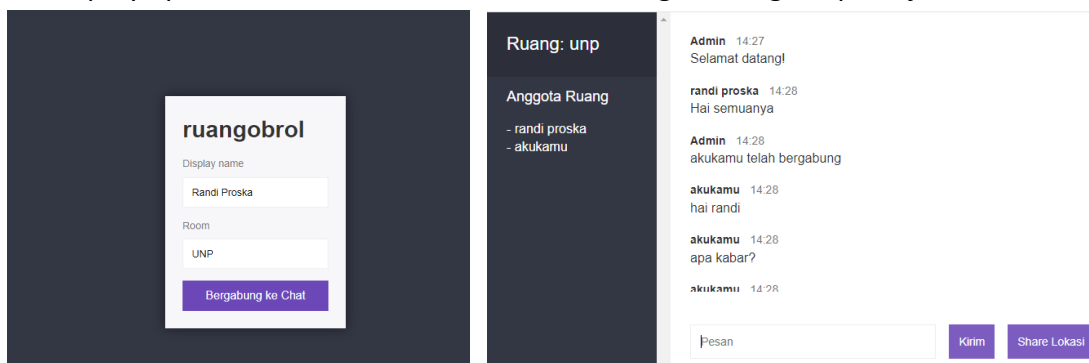
```
socket.emit('join', {username, room}, (error) => {  
  if(error){  
    alert(error)  
    location.href = '/'  
  }  
})
```

8. Silakan download file berikut untuk file-file yang terkait folder **css** dan **img**.

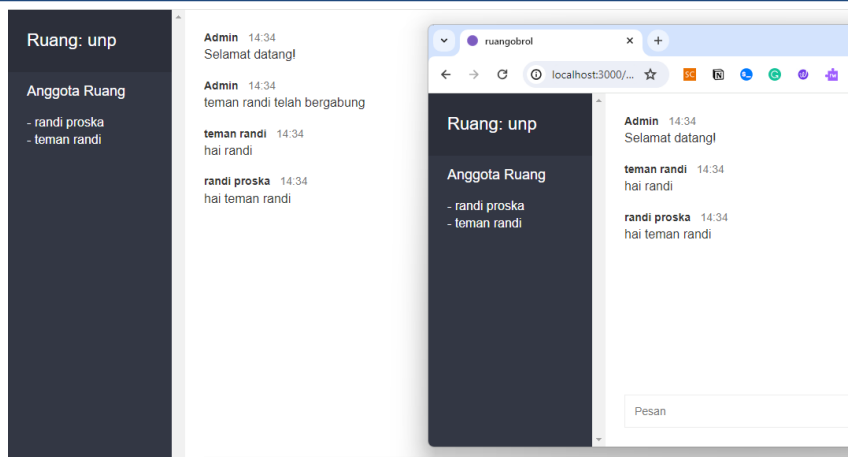
<https://drive.google.com/file/d/1s41FoDuS4djiM2GyeCJwcwTYvmY9PDZU/view?usp=sharing>



9. Perintah untuk menjalankan aplikasi adalah **npm run dev**
10. **PENTING!** Sebelum menjalankan aplikasi, perlu diperhatikan bahwa anda harus menginstall berbagai library yang dibutuhkan, **perhatikan** source code pada file **index.js** di folder **src**. **Salah satu** library utama adalah **socket.io** <https://www.npmjs.com/package/socket.io>
11. Jika telah selesai dan aplikasi berjalan seperti gambar berikut ini, maka lakukanlah commit dan deploy aplikasi anda ke GitHub. Perhatikan langkah-langkah pada jobsheet sebelumnya!



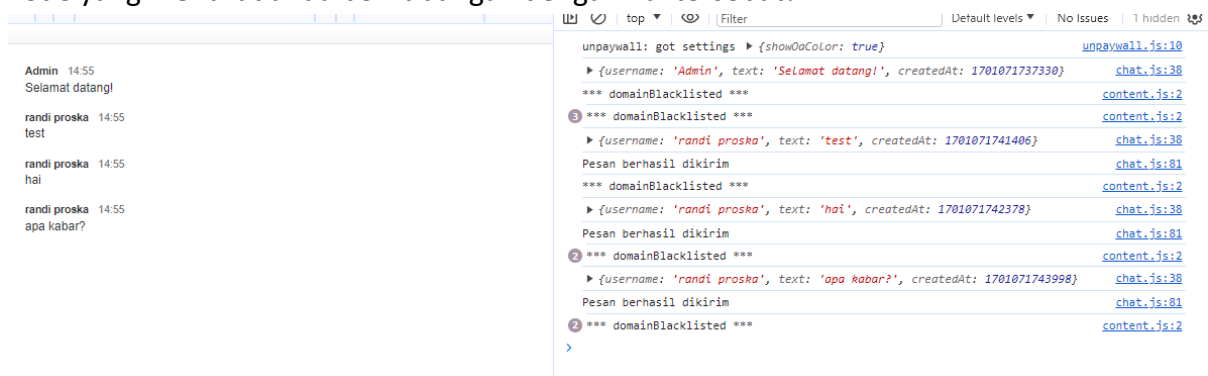
12. Cobalah untuk menjalankan aplikasi dengan mengetikkan **localhost:3000** pada dua tab berbeda atau melalui **browser berbeda** seperti gambar berikut ini!. Gunakan **display name** yang **berbeda** ketika login, namun **nama room tetap sama**.



13. Lakukan kembali langkah 12 namun dengan **nama room yang berbeda**

TUGAS

1. Jelaskan dengan disertai penjelasan baris kode terkait **perbedaan** fungsi **socket.on** yang ada pada file **index.js** di folder **src** dan file **chat.js** pada folder **public/js**!
2. Pada saat anda melakukan proses chat seperti pada langkah 12 dan 13. Bukalah **inspect** pada browser anda. Lalu bukalah menu **console**. Lakukanlah proses chat dan **investigasi** apa yang ditampilkan pada console tersebut. Uraikan penjelasan anda dengan mengaitkannya ke baris kode yang menurut anda berhubungan dengan hal tersebut!



3. Pada file **chat.html** dibagian akhir pada baris kode **<script>** terdapat penggunaan library **mustache**, **moment** dan **qs**. Jelaskan bagaimana ketiga library ini berfungsi dalam aplikasi yang anda buat, kaitkanlah dengan baris kode yang menurut anda berhubungan!
4. Bukalah **chat.js**, dan perhatikan bahwa ada beberapa baris kode yang telah ditandai dengan komentar **elements**, **templates** dan **options**. Jelaskan baris kode tersebut dan bagaimana kode tersebut berhubungan dengan file **chat.html** dan file **index.html**!
5. Jelaskan fungsi file **messages.js** dan **users.js** dan bagaimana baris kode pada kedua file ini terhubung dengan **index.js**, **chat.js** dan kedua file html (**chat.html** dan **index.html**)

6. Bagaimana aplikasi ini bisa mengirimkan lokasi? Jelaskan apa yang terjadi dengan disertai penjelasan baris kode!
7. Kenapa aplikasi ini dijalankan menggunakan perintah **npm run dev** bukan menggunakan perintah node diikuti nama file seperti pada jobsheet-jobsheet sebelumnya? Coba juga jalankan aplikasi menggunakan perintah **npm run start**, investigasi apa yang terjadi dan apa yang membedakannya dengan **npm run dev**?
8. Selain **socket.on**, fungsi socket apa lagi yang digunakan dalam aplikasi ini. Silakan telusuri dan jelaskan pendapat anda disertai dengan baris kode!
9. Jelaskan terkait ini **real-time bidirectional event-based communication** disertai penjelasan baris kode sesuai aplikasi yang anda buat!

REFERENCES

1. Mead, A., & Percival, R. (2019, March). The Complete Node.js Developer Course (3rd ed.) [Oreilly.com]. Packt Publishing.
2. Stevens, R. W., Fenner, B., Rudoff, A. M., & Stevens, W. R. (2003, November 14). UNIX Network Programming: The Sockets Networking API.
<https://doi.org/10.1604/9780131411555>
3. tutorialspoint.com. (n.d.). *WebSockets Tutorial*. Tutorialspoint. Retrieved November 28, 2023, from <https://www.tutorialspoint.com/websockets/index.htm>
4. Mistic, D. (2018, September 13). *A Beginner's Guide to WebSockets*. YouTube Channel - freeCodeCamp.org. Retrieved November 28, 2023, from <https://www.youtube.com/watch?v=8ARodQ4Wlf4>
5. Socket.IO. (2023, November 17). *Introduction / Socket.IO*. Retrieved November 28, 2023, from <https://socket.io/docs/v4/>
6. Mozilla MDN Web. (2023, November 22). *The WebSocket API (WebSockets) - Web APIs / MDN*. MDN Web Docs. Retrieved November 28, 2023, from https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API