SKRIPSI

PEMANFAATAN SMARTPHONE SEBAGAI PENGENDALI PERMAINAN BERBASIS WEB

Priambodo Pangestu

NPM: **2013730055**

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN

«tahun»

UNDERGRADUATE THESIS

«JUDUL BAHASA INGGRIS»

Priambodo Pangestu

NPM: 2013730055

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY

«tahun»

LEMBAR PENGESAHAN

PEMANFAATAN SMARTPHONE SEBAGAI PENGENDALI PERMAINAN BERBASIS WEB

Priambodo Pangestu

NPM: 2013730055

Bandung, «tanggal» «bulan» «tahun»

Menyetujui,

Pembimbing Utama	Pembimbing Pendamping
Pascal Alfadian, M.Comp.	

Ketua Tim Penguji	Anggota Tim Penguji
«penguji 1»	**«penguji 2»**

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PEMANFAATAN SMARTPHONE SEBAGAI PENGENDALI PERMAINAN BERBASIS WEB

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal «tanggal» «bulan» «tahun»

Meterai
Rp. 6000

Priambodo Pangestu
NPM: 2013730055

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»
Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini. . . ?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini . . . »
Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Bandung, «bulan» «tahun»

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

WebSockets adalah teknologi yang memungkinkan *web browser* pengguna dan *web server* membuka sesi komunikasi interaktif satu sama lain. Teknologi *WebSockets* didesain untuk diimplementasikan pada *web browser* dan *web server*, tetapi dapat juga digunakan oleh setiap aplikasi *client* maupun *server*. *WebSockets* memiliki standar yang menyediakan cara agar *web server* dapat mengirim konten ke *web browser* tanpa diminta oleh *client*, dan memungkinkan agar pesan dikirimkan berulang-ulang dengan tetap menjaga koneksi yang terbuka. Oleh karena itu, protokol *WebSockets* memungkinkan interaksi antara *web browser* dan *web server* dengan *overhead* yang rendah, dan juga memfasilitasi transfer data *realtime* dari *server* maupun menuju *server*.

Salah satu teknologi yang memanfaatkan protokol *WebSockets* adalah *Socket.io*. Teknologi ini memungkinkan untuk melakukan komunikasi secara *realtime*, dan dua arah antara *client* dan *server*. Socket.io memiliki dua bagian: *client-side library* yang berjalan didalam *web browser*, dan *server-side library* yang berjalan pada *Node.js*. *Socket.io* memiliki fitur-fitur yang beragam, seperti melakukan broadcast ke beberapa *sockets*, dan menyimpan data yang berhubungan dengan masing-masing *client*. Teknologi ini sangat berguna untuk membantu membangun sebuah aplikasi yang membutuhkan koneksi *realtime* seperti dalam aplikasi *chatting* maupun *game*.

Untuk memanfaatkan protokol *WebSockets* dalam membangun aplikasi permainan, akan dibutuhkan beberapa teknologi yang dapat membantu pembangunan aplikasinya. Salah satu teknologi tersebut yaitu *Canvas API*. Teknologi ini merupakan bagian dari *HTML5 element* yang dapat digunakan untuk menggambar suatu grafis melalui *JavaScript* secara *on the fly*. *Canvas API* dapat juga digunakan untuk membuat komposisi foto, membuat animasi, dan membuat *real-time video processing* atau *rendering*. Oleh karena itu, fungsi-fungsi yang ada pada *Canvas API* akan membantu pembangunan aplikasi permainan terutama pada bagian pengembangan grafis pada aplikasinya.

Teknologi lain yang dapat membantu membangun aplikasi permainan dalam memanfaatkan protokol *WebSockets* yaitu *Node.js*. Teknologi ini merupakan sebuah *platform* yang didesain untuk mengembangkan aplikasi berbasis web pada bagian *web server*. *Node.js* ditulis dalam sintaks bahasa pemrograman *JavaScript* dan menggunakan *V8* yang merupakan *engine JavaScript* milik perusahaan *Google* untuk mengeksekusi *JavaScript* pada *web server*. *Node.js* memiliki sifat *non-blocking*, yang berarti *Node.js* tidak akan menunggu untuk mengerjakan *request* selanjutnya. *Node.js* pun sangat cepat dalam mengeksekusi suatu kode karena menggunakan *engine JavaScript V8*. Fitur-fitur yang dimiliki oleh *Node.js* akan sangat membantu untuk membangun aplikasi permainan yang membutuhkan koneksi *real-time*.

Pada skripsi ini, akan dibuat sebuah aplikasi permainan yang memanfaatkan protokol *WebSockets*, dimana dalam penggunaan protokol tersebut akan dibantu dengan teknologi *Socket.io*. Selain itu, aplikasi yang dibuat akan memanfaatkan *personal computer (PC)* dan *smartphone* untuk pengembangan aplikasinya. Para pemain akan mengkoneksikan *smartphone* miliknya pada suatu *PC*, dimana *smartphone* tersebut akan berfungsi sebagai *controller* untuk memainkan permainannya. Oleh karena itu, protokol *WebSockets* akan digunakan sebagai koneksi antara *smartphone* dan *PC*

dalam aplikasi permainan yang akan dibangun. Aplikasi permainan akan menggunakan teknologi berbasis web, sehingga untuk memainkannya, *client* bisa mengakses melalui *web browser* tanpa harus berada di satu jaringan lokal yang sama.

1.2 Rumusan Masalah

1. Bagaimana membangun aplikasi permainan berbasis web dengan memanfaatkan protokol *WebSockets* untuk penggunaan *smartphone* sebagai pengendali permainan berbasis web ?
2. Berapa *latency* yang dihasilkan berdasarkan penggunaan protokol *WebSockets* ?

1.3 Tujuan

1. Mengetahui cara membangun aplikasi permainan berbasis web dengan memanfaatkan protokol *WebSockets* untuk penggunaan *smartphone* sebagai pengendali permainan berbasis web.
2. Mengetahui jumlah *latency* yang dihasilkan berdasarkan pemanfaatan protokol *WebSockets*.

1.4 Batasan Masalah

Batasan masalah yang dibuat terkait dengan pengerjaan skripsi ini adalah sebagai berikut:

1. Aplikasi permainan yang dibuat merupakan permainan *multiplayer* yang hanya bisa dimainkan oleh dua orang saja.

1.5 Metodologi

Metodologi yang dilakukan dalam pengerjaan skripsi ini adalah sebagai berikut:

1. Studi literatur mengenai :
   - *WebSockets* yang akan digunakan untuk koneksi antara *smartphone* dan *PC*.
   - *Socket.io* sebagai teknologi yang akan menggunakan *WebSockets* dalam pembangunan aplikasi.
   - *Canvas API* yang akan digunakan untuk antarmuka permainan.
   - *Node.js* sebagai *web server* dalam pembangunan aplikasi.
   - *Express.js* sebagai *Node.js framework* yang akan digunakan untuk mengatur penyimpanan data dalam *Node.js*
2. Menganalisis aplikasi sejenis.
3. Merancang antarmuka permainan pada *PC* dan *smartphone*. Antarmuka pada *PC* akan berbeda dengan yang ada di *smartphone*, karena *smartphone* akan bekerja sebagai *controller* dan *PC* akan bekerja sebagai *console*.
4. Menyusun cara bermain aplikasi permainan yang dibangun.
5. Mengimplementasi program aplikasi permainan berbasis web.
6. Menganalisis *latency* yang dihasilkan pada aplikasi.
7. Melakukan eksperimen dan pengujian yang melibatkan responden.

1.6 Sistematika Pembahasan

Setiap bab dalam skripsi ini memiliki sistematika penulisan yang dijelaskan kedalam poin-poin sebagai berikut:

1. Bab 1 : Pendahuluan
   Membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.

2. Bab 2 : Dasar Teori
   Membahas mengenai teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang *WebSockets*, *Socket.io*, *Node.js*, *Express.js*, dan *Canvas API*.

3. Bab 3 : Analisis
   Membahas mengenai analisa masalah.

4. Bab 4 : Perancangan
   Membahas mengenai perancangan yang dilakukan sebelum melakukan tahapan implementasi.

5. Bab 5 : Implementasi dan Pengujian
   Membahas mengenai implementasi dan pengujian yang telah dilakukan.

6. Bab 6 : Kesimpulan dan Saran
   Membahas hasil kesimpulan dari keseluruhan penelitian ini dan saran-saran yang dapat diberikan untuk penelitian berikutnya.

BAB 2

LANDASAN TEORI

Pada bab ini akan dijelaskan landasan teori mengenai *WebSockets*, *Socket.io*, *Node.js*, *Express.js*, dan *Canvas API*.

2.1 WebSockets

WebSockets merupakan *Application Programming Interface (API)* yang memiliki kemampuan untuk membuka sesi komunikasi interaktif antara *browser* pengguna dan *server* [1]. Dengan *API* ini, pengguna dapat mengirim pesan ke *server* dan menerima respon tanpa harus melakukan *polling* pada *server* terlebih dahulu.

Subbab-subbab berikut menjelaskan kelas-kelas yang ada pada *WebSockets*.

2.1.1 WebSocket

Kelas ini merupakan inti untuk mengakses fungsi yang ada pada *WebSockets*. Sebuah objek *WebSocket* dapat membuat dan mengelola koneksi *WebSocket* ke server, serta dapat mengirim dan menerima data pada koneksi tersebut.

Berikut merupakan konstruktor dari kelas *WebSocket*:

```
WebSocket WebSocket(in DOMString url, in optional DOMString protocols);
```

- **url**, parameter wajib yang menunjukan *URL* mana yang akan direspon oleh *WebSocket server*.
- **protocols**, parameter pilihan (tidak harus ada pada parameter) yang dapat berupa satu *string* atau *array of strings*. Parameter *protocols* merepresentasikan nama dari subprotokol yang akan digunakan oleh objek *WebSocket*. Apabila subprotokol tersedia pada parameter, maka *server* akan memeriksa apakah subprotokol tersebut dapat diterima atau tidak. *Server* akan memberikan respon apabila subprotokol dapat diterima, dan akan menghasilkan suatu *error* apabila tidak dapat diterima. Contoh subprotokol yang dapat digunakan yaitu:
  - **chat**
  - **superchat**

Konstruktor dari kelas *WebSocket* dapat menampilkan suatu *exception* seperti berikut:

```
SECURITY_ERR
```

Exception tersebut menandakan bahwa *port* yang akan digunakan untuk melakukan koneksi diblokir.

Atribut yang dimiliki oleh kelas *WebSocket* yaitu:

- **binaryType**
  tipe: **DOMString**
  Sebuah *string* yang menandakan tipe dari data biner yang dikirimkan oleh koneksi tertentu. Nilai dari atribut ini dapat berupa *"ArrayBuffer"* apabila objek dari *ArrayBuffer* digunakan.

- **bufferedAmount**
  tipe: **unsigned long**
  Jumlah *bytes* dari data yang belum dikirimkan oleh *method* **send()**. Nilai dari atribut ini akan kembali menjadi nol apabila seluruh data sudah dikirimkan. Apabila koneksi terputus, nilai atribut ini tidak akan kembali menjadi nol dan akan tetap bertambah apabila terus dilakukan pemanggilan pada *method* **send()**.

- **onclose**
  tipe: **EventListener**
  Event listener yang dipanggil saat atribut *readyState* dalam koneksi *WebSocket* berubah menjadi *CLOSED*. *Listener* akan menerima objek dari *CloseEvent* dengan nilai *"close"*.

- **onerror**
  tipe: **EventListener**
  Event listener yang dipanggil saat terjadi *error*. *Event* tersebut akan bernilai *"error"*.

- **onmessage**
  tipe: **EventListener**
  Event listener yang dipanggil saat atribut *readyState* dalam koneksi *WebSocket* berubah menjadi *OPEN*. Hal tersebut menandakan bahwa koneksi sudah siap untuk mengirim dan menerima data. *Event* tersebut akan bernilai *"open"*.

- **protocol**
  tipe: **DOMString**
  String yang menandakan sebuah nama dari sub-protokol yang dipilih oleh *server*. Atribut ini akan menjadi salah satu masukan parameter yang dibutuhkan untuk konstruksi kelas *WebSocket*.

- **readyState**
  tipe: **unsigned short**
  Menunjukan kondisi koneksi saat ini. Atribut ini memiliki beberapa konstanta yang menunjukan kondisi dari koneksi *WebSocket*. Konstanta tersebut sebagai berikut:

  - **CONNECTING**
    nilai: 0
    Koneksi belum terbuka.
  - **OPEN**
    nilai: 1
    Koneksi sudah terbuka dan siap untuk melakukan komunikasi.
  - **CLOSING**
    nilai: 2
    Koneksi sedang dalam proses menutup.
  - **CLOSED**
    nilai: 3
    Koneksi sudah tertutup atau tidak dapat dibuka.

- **url**
  tipe: **DOMString**
  URL yang akan dituju oleh objek *WebSocket*. Atribut ini akan menjadi salah satu masukan parameter untuk konstruksi kelas *WebSocket*.

Kelas *WebSocket* memiliki dua buah *method*, yaitu:

- **void close(in optional unsigned long code, in optional DOMString reason)**
  Berfungsi untuk menutup suatu koneksi atau menghentikan proses koneksi.
  Parameter:
  - **code** nilai numerik yang menunjukan kode status, yang menjelaskan mengapa suatu koneksi ditutup. Apabila parameter ini tidak tersedia, maka akan diasumsikan dengan nilai *default* yaitu 1000 yang berarti transaksi selesai.
  - **reason** *string* yang menjelaskan mengapa suatu koneksi ditutup.

  Method ini dapat melemparkan eksepsi seperti berikut:
  - **INVALID_ACCESS_ERR** parameter *code* yang tidak valid.
  - **SYNTAX_ERR** parameter *reason* yang melebihi batas yang telah ditentukan.

- **void send(in DOMString data)**
  Berfungsi untuk mengirimkan data ke *server* melalui koneksi *WebSocket*, dan menambah nilai dari *bufferedAmount* sebanyak jumlah *bytes* yang dibutuhkan untuk menampung data.
  Parameter
  Tipe data yang dikirimkan pada parameter dapat berbeda-beda, Beberapa tipe tersebut yaitu sebagai berikut:
  - **USVString** sebuah teks *string* yang ditambahkan ke *buffer* dalam format *UTF-8*. Nilai dari *bufferedAmount* akan bertambah sesuai dengan jumlah *bytes* yang dibutuhkan untuk menyimpan *UTF-8 string*.
  - **ArrayBuffer** data biner yang disimpan pada *fixed-length buffer*, dimana objek dari *ArrayBuffer* dimanipulasi oleh objek *TypedArray*.

  Method ini dapat melemparkan eksepsi seperti berikut:
  - **INVALID_STATE_ERR** koneksi saat ini tidak terbuka.
  - **SYNTAX_ERR** parameter *data* tidak valid.

2.1.2 CloseEvent

Kelas ini akan menangani koneksi *WebSocket* yang ditutup. Objek *CloseEvent* akan dikirim ke *client* saat koneksi ditutup. Objek tersebut akan dikirimkan ke *listener* yang ditunjukan oleh atribut *onclose* milik objek *WebSocket*.

Konstruksi kelas ini yaitu:

- **new CloseEvent(typeArg, closeEventInit);**
  Parameter:
  - **typeArg**
    tipe: **DOMString**
    nama dari suatu *event* yang akan dikirimkan.
  - **closeEventInit** bersifat pilihan, dan memiliki beberapa nilai sebagai berikut:
    * *"wasClean"*
      tipe: **boolean**
      menunjukan apakah koneksi sudah ditutup dengan baik atau belum.
    * *"code"*
      tipe: **unsigned short**
      kode status yang menunjukan mengapa koneksi ditutup.

  - *"reason"*
    tipe: **DOMString**
    teks yang menunjukan alasan mengapa koneksi ditutup oleh *server*.

Berikut merupakan nilai-nilai dari kode status koneksi ditutup:

- **0-999**
  nama: -
  Reserved. Tidak digunakan.

- **1000**
  nama: **Normal Closure**
  Penutupan normal, yang berarti koneksi sudah menyelesaikan apapun tujuan dari koneksi tersebut.

- **1001**
  nama: **Going Away**
  Endpoint menghilang karena kesalahan server atau *browser* tidak lagi mengakses halaman yang sudah membuka koneksi.

- **1002**
  nama: **Protocol Error**
  Endpoint menghentikan koneksi karena adanya kesalahan protokol.

- **1003**
  nama: **Unsupported Data**
  Koneksi dihentikan karena *endpoint* menerima data dengan tipe yang tidak bisa diterima (contoh: *text-only endpoint* menerima data biner).

- **1004**
  nama: -
  Reserved. Makna dari kode tersebut akan dijelaskan di waktu yang akan datang.

- **1005**
  nama: **No Status Recieved**
  Reserved. Menandakan bahwa tidak ada kode status yang tersedia.

- **1006**
  nama: **Abnormal Closure**
  Reserved. Menandakan bahwa koneksi ditutup secara tidak normal (contoh: tidak ada *close frame* yang dikirimkan).

- **1007**
  nama: **Invalid frame payload data**
  Endpoint menghentikan koneksi karena pesan yang diterima berisi data yang tidak konsisten (contoh: data *non-UTF-8* berada di dalam pesan teks).

- **1008**
  nama: **Policy Violation**
  Endpoint menghentikan koneksi karena menerima pesan yang melanggar kebijakan. Kode status ini dapat digunakan apabila tidak ada kode status lain yang cocok atau digunakan untuk tidak menunjukan kebijakan lebih rinci.

- **1009**
  nama: **Message too big**
  Endpoint menghentikan koneksi karena menerima *frame* data yang terlalu besar.

- **1010**
  nama: **Missing Extension**
  Client menghentikan koneksi karena *server* tidak menangani satu atau beberapa ekstensi yang diminta oleh *client*.

- **1011**
  nama: **Internal Error**
  Server menghentikan koneksi karena mengalami kondisi tertentu yang menyebabkan tidak bisa memenuhi permintaan *client*.

- **1012**
  nama: **Service Restart**
  Server menghentikan koneksi karena harus mengulang kembali koneksi.

- **1013**
  nama: **Try Again Later**
  Server menghentikan koneksi karena ada kondisi yang harus ditangani untuk sementara (contoh: *overloaded*).

- **1014**
  nama: **Bad Gateway**
  Server bertindak sebagai *gateway* atau *proxy* dan menerima respon yang tidak benar dari *upstream server*.

- **1015**
  nama: **TLS Handshake**
  Reserved. Menandakan bahwa koneksi ditutup karena gagal melakukan *TLS handsake* (contoh: sertifikat *server* tidak dapat diverifikasi).

- **1016-1999**
  nama: -
  Reserved. Akan digunakan oleh standar *WebSocket* di waktu yang akan datang.

- **2000-2999**
  nama: -
  Reserved. Akan digunakan oleh ekstensi *WebSocket*.

- **3000-3999**
  nama: -
  Tersedia untuk digunakan oleh *libraries* dan *frameworks*.

- **4000-4999**
  nama: -
  Tersedia untuk digunakan oleh aplikasi.

2.1.3 MessageEvent

Kelas ini merepresentasikan pesan yang diterima oleh suatu objek tertentu. *Constructor* dari kelas ini yaitu:

```
MessageEvent()
```

Beberapa properti yang dimiliki oleh kelas ini yaitu:

- **MessageEvent.data**
  Merupakan data yang telah dikirimkan oleh pengirim.

- **MessageEvent.lastEventId**
  Merepresentasikan *ID* yang unik untuk sebuah *Event*.

Contoh penggunaan dari beberapa properti tersebut sebagai berikut:

```
//MessageEvent.data
myWorker.onmessage = function(e) {
result.textContent = e.data;
console.log('Message received from worker');
};

//MessageEvent.lastEventId
myWorker.onmessage = function(e) {
result.textContent = e.data;
console.log('Message received from worker');
console.log(e.lastEventId);
};
```

2.2 Socket.io

Socket.io merupakan salah satu teknologi yang memanfaatkan protokol *WebSockets* [2]. Teknologi ini memungkinkan sebuah aplikasi untuk melakukan komunikasi dua arah secara *real-time*. *Socket.io* dapat dijalankan di setiap *platform, browser*, dan gawai.

Socket.io dibagi menjadi dua *API*, yaitu *Server API* dan *Client API*. Subbab-subbab berikut menjelaskan kelas-kelas yang dimiliki *Socket.io*.

2.2.1 Server API

Kelas-kelas yang ada pada *Server API* digunakan untuk menangani proses yang terjadi dalam *server*[3]. Kelas-kelas tersebut adalah sebagai berikut:

1. **Server**
   Kelas ini merupakan inti untuk dapat menangani proses yang terjadi dalam *socket.io server*. Kelas ini memiliki tiga konstruktor seperti berikut:

   - **new Server(httpServer[, options])**
     Parameter:
     - **httpServer**
       tipe: **http.Server**
       Server yang akan dituju.
     - **options**
       tipe: **Object**
       Parameter ini dapat berupa berbagai jenis objek. Objek-objek tersebut yaitu sebagai berikut:
       * **path**
         tipe: **String**
         Nama dari path yang akan ditangkap oleh *server* (contoh: */socket.io*).
       * **serveClient**
         tipe: **Boolean**
         Menunjukan apakah *server* akan melayani *file* dari *client* atau tidak.

Untuk dapat menggunakan fitur yang ada pada *socket.io*, harus menambahkan modul *socket.io* pada konstanta tertentu. Hal tersebut dapat dilakukan dengan dua cara, yaitu menggunakan kata kunci *new* atau tanpa menggunakan kata kunci *new*:

- Menggunakan *new*

```
const Server = require('socket.io');
const io = new Server();
```

- Tanpa menggunakan *new*

```
const io = require('socket.io')();
```

Contoh implementasi konstruktor:

```
const Server = require('socket.io');
const http = require('http').createServer();

const io = new Server(http, {
        path: '/test',
        serveClient: false
});
```

- **new Server(port[,options])**
  Parameter:
  - **port**
    tipe: **Number**
    Nomor *port* yang akan dituju.
  - **options**
    tipe: **Object**
    Sama seperti konstruktor pertama, parameter ini dapat berupa berbagai jenis objek.

  Contoh pemakaian konstruktor:

```
const Server = require('socket.io');
const io = new Server(3000, {
        path: '/test',
        serveClient: false
});
```

- **new Server(options)**
  Parameter:
  - **options**
    tipe: **Object**
    Sama seperti konstruktor pertama, parameter ini dapat berupa berbagai jenis objek.

  Contoh implementasi konstruktor:

```
const Server = require('socket.io');
const io = new Server({
        path: '/test',
        serveClient: false
});
```

Beberapa *method* yang dimiliki oleh kelas ini yaitu sebagai berikut:

- **server.serveClient([value])**
  Parameter:

  - **value**
    tipe: **Boolean**

  Kembalian: *Server* atau *Boolean*.
  Apabila parameter *value* bernilai *true*, maka *server* akan menangani *file* dari *client*. Apabila tidak ada argumen pada *method* ini, maka kembalian akan berupa status *default* dari *serveClient* saat ini (*true*).

- **server.path([value])**
  Parameter:
  - **value**
    tipe: **String**

  Kembalian: *Server* atau *String*
  Parameter *value* akan menunjukan nilai dari *path* yang akan dituju. Secara *default* nilai dari *path* akan diisi dengan */socket.io*. Apabila tidak ada argumen pada *method* ini, maka kembalian akan berupa nilai dari *value* saat ini.
- **server.attach(httpServer[, options])**

2. **Namespace**
3. **Socket**
4. **Client**

Namespace

Kelas ini merepresentasikan kumpulan dari *sockets* yang terhubung dalam lingkup tertentu, yang diidentifikasi oleh sebuah *pathname*. *Client* selalu terhubung ke */* (*namespace* utama), kemudian dapat terhubung ke *namespace* lainnya ketika menggunakan koneksi yang sama.

Beberapa properti yang dimiliki oleh kelas ini yaitu:

- **namespace.name**
  Sebuah *string* yang merupakan *identifier* pada *namespace*.
- **namespace.connected**
  Sebuah *hash* dari objek *Socket* yang terhubung pada *namespace* saat ini.
- **namespace.adapter**
  Sebuah adaptor yang digunakan untuk *namespace* tertentu.

Beberapa *method* yang dimiliki oleh kelas ini yaitu:

- **namespace.emit(eventName[, ...args])**
  Akan menyebarkan suatu *event* ke semua *clients* yang sedang melakukan koneksi.
- **namespace.client(callback)**
  Akan mendapatkan daftar para *clients* yang sedang terkoneksi dengan *namespace* ini.

Berikut merupakan contoh implementasi dari beberapa *method* tersebut:

```
//namespace.emit
const io = require('socket.io')();
io.emit('an event sent to all connected clients');

//namespace.client
const io = require('socket.io')();
```

```
io.of('/chat').clients((error, clients) => {
if (error) throw error;
console.log(clients); // => [PZDoMHjiu8PYfRiKAAAF, Anw2LatarvGVVXEIAAAD]
});
```

Socket

Kelas ini merupakan kelas yang sangat mendasar untuk melakukan interaksi dengan *browser* milik *client*. Sebuah *Socket* dimiliki oleh *Namespace* tertentu (secara *default* menggunakan /).

Beberapa properti yang dimiliki oleh kelas ini yaitu :

- **socket.id**
  Merupakan tanda pengenal yang unik untuk sesi tertentu.
- **socket.client**
  Merupakan *reference* ke objek *Client* tertentu.
- **socket.request**
  Merupakan *getter proxy* yang mengembalikan referensi ke *request* yang berasal dari *Client* tertentu.

Beberapa *method* yang dimiliki oleh kelas ini yaitu :

- **socket.send([...args[, ack]])**
  Berfungsi untuk mengirimkan pesan tertentu.
- **socket.emit(eventName[, ...args][, ack])**
  Berfungsi untuk mengeluarkan suatu *event* kepada *socket* yang diidentifikasi oleh nama *event* tersebut.
- **socket.disconnect(close)**
  Berfungsi untuk mengakhiri koneksi milik *client* saat ini.

Berikut merupakan contoh implementasi dari beberapa *method* tersebut:

```
//emit
socket.emit('hello', 'world');

//disconnect
io.on('connection', (socket) => {
setTimeout(() => socket.disconnect(true), 5000);
});
```

Client

Kelas ini merepresentasikan koneksi *transport* yang masuk[4]. *Client* dapat terhubung dengan beberapa *Sockets* yang termasuk dalam *Namespaces* yang berbeda.

Atribut-atribut yang dimiliki oleh kelas ini yaitu:

- **client.conn**
  Merupakan referensi kepada koneksi *Socket*.
- **client.request**
  Berfungsi untuk mengakses *header* dari *request* seperti *Cookie* atau *User-Agent*.

2.2.2 Client API

Kelas-kelas yang ada pada *Client API* digunakan untuk menangani proses-proses yang terjadi pada bagian *client*.

Kelas-kelas yang ada pada *Client API* yaitu:

Manager

Kelas ini memiliki *constructor* sebagai berikut:

```
Manager(url[, options]
```

- **url** , merupakan sebuah *string* yang merepresentasikan suatu *url* yang akan dituju.
- **options**, merupakan suatu objek yang dapat berupa :
  - **path**, merupakan sebuah *string* yang merepresentasikan suatu *path* yang akan dituju dalam bagian *server*.
  - **reconnection**, merupakan sebuah *boolean* yang menunjukan apakah dapat melakukan koneksi ulang secara otomatis atau tidak.
  - **timeout**, merupakan angka yang menunjukan koneksi sudah mencapai *timeout* sebelum terjadi error pada koneksi.

Socket

Kelas ini memiliki atribut **id**, yang merupakan tanda pengenal unik untuk sesi saat ini.

Beberapa *method* yang dimiliki oleh kelas *Socket* yaitu:

- **socket.open()**
  Berfungsi untuk membuka suatu koneksi *socket* tertentu.
- **socket.emit(eventName[, ...args][, ack])**
  Berfungsi untuk mengeluarkan suatu *event* kepada *socket* yang diidentifikasi oleh nama *event* tersebut.
- **socket.close()**
  Berfungsi untuk menutup koneksi suatu *socket* secara manual.

2.3 Node.js

Node.js adalah *JavaScript runtime* yang dibangun berdasarkan *V8* yang merupakan *JavaScript engine* milik perusahaan *Google* [5]. *Node.js* memiliki model *event-driven*, dan *non-blocking I/O* yang membuat teknologi tersebut efisien dalam implementasinya. Teknologi ini menyediakan beberapa kelas yang berfungsi untuk mengimplementasi fitur-fitur yang dimiliki.

Beberapa kelas yang terdapat pada *Node.js* yaitu sebagai berikut:

2.3.1 Console

Console merupakan perangkat *debugging* yang memiliki persamaan dengan mekanisme *JavaScript* yang disediakan oleh *web browsers*. Kelas ini dapat digunakan untuk membuat *logger* sederhana dengan *output streams* yang dapat dikonfigurasi. Agar dapat menggunakan kelas ini, dapat dilakukan langkah berikut:

```
const Console = require('console').Console;
```

Salah satu *method* yang dimiliki oleh kelas ini yaitu:

- **console.log()**
  Akan mengeluarkan beberapa argumen pada *stdout (standard output).*

Berikut merupakan contoh implementasi dari *method* tersebut:

```
const count = 5;
console.log('count: %d', count);
// Prints: count: 5, to stdout
```

2.3.2 File System

Modul ini berfungsi untuk menangani proses pengaturan *file* pada *server*. Untuk dapat menggunakan modul ini diperlukan **require('fs')** agar dapat mengakses fitur-fitur yang ada.

Salah satu kelas yang terdapat pada *File System* yaitu:

fs.WriteStream

Kelas ini memiliki beberapa *method* sebagai berikut:

- **fs.appendFile(file, data[, options],callback)**
  Berfungsi untuk menambahkan suatu data kedalam *file*, dan membuat *file* baru apabila *file* yang dituju belum tersedia.
- **fs.readFile(file[, options], callback)**
  Berfungsi untuk membaca seluruh konten dari suatu *file*.

Berikut merupakan contoh implementasi dari beberapa *method* tersebut:

```
//appendFile
fs.appendFile('message.txt', 'data to append', (err) => {
if (err) throw err;
console.log('The "data to append" was appended to file!');
});

//readFile
fs.readFile('/etc/passwd', (err, data) => {
if (err) throw err;
console.log(data);
});
```

2.3.3 HTTP

Interfaces HTTP pada *Node.js* digunakan untuk menangani *request* dari protokol *HTTP* yang secara *native* sulit untuk digunakan. *Interface* ini akan menangani protokol *HTTP* dengan tidak melakukan *buffer* pada seluruh *request* atau *responses*.

Berikut akan dijelaskan kelas-kelas yang ada pada *interface HTTP*.

1. **http.IncomingMessage**
   Objek dari kelas ini akan dibuat oleh kelas *http.Server* atau *http.ClientRequest* dan memasukannya sebagai argumen suatu *event 'request'* dan *'response'*. Objek tersebut dapat digunakan untuk mengakses status *response*, *headers*, dan data. Kelas ini mengimplementasi *interface Readable Stream*, beserta *method, events,* dan properti yang ada didalamnya.

   Properti:

- **message.headers**
  Kembalian: *headers* milik objek *request/response.*
- **message.rawHeaders**
  Kembalian: bentuk *raw* dari *headers* milik objek *request/response.*
- **message.statusCode**
  Kembalian: tiga dijit kode status *HTTP response.* Contoh: 404.
- **message.statusMessage**
  Kembalian: pesan status *HTTP response* Contoh: *OK* atau *Internal Server Error.*
- **message.url**
  Kembalian: *URL string* yang muncul pada permintaan *HTTP.*

2. **http.ClientRequest**

3. **http.Server**
   Kelas ini merupakan turunan dari *net.Server. Event* yang dimiliki kelas ini yaitu sebagai berikut:

   - **'close'**
     Dipancarkan apabila *server* sudah ditutup.

   Beberapa properti yang dimiliki oleh kelas ini yaitu:

   - **server.listening**
     Mengembalikan *boolean* yang menandakan apakah *server* melakukan proses *listening* untuk suatu koneksi atau tidak.
   - **server.maxHeadersCount**
     Mengembalikan *number* yang menandakan batas maksimum suatu *headers* yang masuk. Nilai default dari properti ini yaitu 2000.
   - **server.timeout**
     Mengembalikan *number* yang menandakan *timeout* dalam milidetik.

   Beberapa *Method* yang dimiliki oleh kelas ini yaitu:

   - **server.listen()**
     Memulai *server HTTP* melakukan proses *listening* untuk suatu koneksi.
   - **server.setTimeout([msecs][,callback])**
     Parameter:
     - **msec** nilai *timeout* dalam milidetik. secara default bernilai 120000 (2 menit).
     - **callback** fungsi *callback.*

     Kembalian: objek *server.*

     Method ini menetapkan nilai *timeout* untuk *sockets* dan memancarkan *event 'timeout'* pada objek *Server.*
   - **server.close([callback])**
     Parameter:
     - **callback** fungsi *callback.*

     Method ini menghentikan *server* untuk menerima koneksi baru.

Beberapa *Method* yang dimiliki oleh *HTTP* yaitu sebagai berikut:

- **http.createServer([requestListener])**
  Parameter:

– **requestListener** fungsi yang akan secara otomatis ditambahkan pada *event 'request'* milik kelas *http.Server*.

Kembalian: objek *http.Server*

Method ini akan membuat objek *http.Server* untuk menangani *request* dari *client* dan memberikan *response* kepada *client*. Fungsi yang diberikan pada *method* ini akan dipanggil satu kali setiap *request* dibuat kepada *server*.

- **http.request(options[,callback])**
  Parameter:

  – **options**
  Dapat berupa *Object, string* atau *URL*. Berikut jenis-jenis *options* yang dapat menjadi parameter:

  * **protocol**
    tipe: **string**
    Protokol yang digunakan.
  * **host**
    tipe: **string**
    Nama domain atau alamat *IP* milik server.
  * **hostname**
    tipe: **string**
    Nama lain untuk *host*.
  * **port**
    tipe: **number**
    Port untuk *server*.
  * **path**
    tipe: **string**
    Path untuk permintaan.
  * **headers**
    tipe: **Object**
    Objek yang berisi permintaan *headers*.
  * **timeout**
    tipe: **number**
    Nomor yang menentukan *timeout* dari suatu *socket* dalam milidetik.

  – **callback**
  tipe: **Function**
  Fungsi *callback*.

  Kembalian: objek dari kelas *http.ClientRequest*.

  Method ini digunakan untuk menangani permintaan *HTTP* pada *server*. Berikut contoh implementasi dari *method* ini:

```
const options = {
        hostname: 'www.google.com',
        port: 80,
        path: '/upload',
        method: 'POST',
        headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Content-Length': Buffer.byteLength(postData)
        }
```

```
};

const req = http.request(options, (res) => {
        console.log(`STATUS: ${res.statusCode}`);
        console.log(`HEADERS: ${JSON.stringify(res.headers)}`);
        res.setEncoding('utf8');
        res.on('data', (chunk) => {
        console.log(`BODY: ${chunk}`);
        });
        res.on('end', () => {
        console.log('No more data in response.');
        });
});

req.end();
```

2.3.4 Events

Node.js dibangun berdasarkan arsitektur *event-driven* dengan sifat *asynchronous*, dimana jenis-jenis objek tertentu akan memancarkan suatu *events* secara berkala dan akan memanggil objek *Function* ("*listeners*").

Semua objek yang memancarkan *events* merupakan turunan dari kelas *EventEmitter*. Objek tersebut akan memanggil *method eventEmitter.on()* yang memungkinkan satu atau beberapa fungsi dapat ditangani dalam suatu *event* yang dipancarkan oleh objek saat ini.

Kelas *EventEmitter* dapat didefinisikan dengan memanggil modul *events* seperti berikut :

```
const EventEmitter = require('events');
```

Sebuah *EventEmitter* akan memancarkan *'event' 'newListener'* pada saat *listeners* baru akan ditambahkan, dan *'removeListener'* akan dipancarkan saat *listeners* saat ini akan dihapus.

Berikut merupakan beberapa *method* yang dimiliki oleh kelas *EventEmitter*:

- **eventEmitter.on(eventName, listener)**
  Parameter:
  - *eventName*, nama dari suatu *event* yang akan dipancarkan.
  - *listener*, suatu fungsi *callback* yang akan menangani *event* dari *eventName*.

  Kembalian: referensi kepada *EventEmitter*.

  Method ini berfungsi untuk mencatat suatu listener yang akan digunakan. Fungsi *listener* yang menjadi parameter *method* ini akan ditambahkan ke *index* terakhir dari *array of listeners* pada *eventName*. Tidak akan ada pengecekan apakah fungsi *listener* sudah dimasukan sebelumnya. Oleh karena itu, pemanggilan *eventName* dan *listener* secara berulang akan menyebabkan fungsi *listener* dimasukan kedalam *array* dan dipanggil secara berulang. *Method* ini juga akan mengembalikan *reference* kepada *EventEmitter*, sehingga pemanggilan dapat saling menyambung dengan pemanggilan lainnya.

  Berikut merupakan contoh implementasi dari *method* ini:

```
const EventEmitter = require('events');

class MyEmmit extends EventEmitter {}

const myEmmit = new MyEmitter();
```

```
myEmmit.on('event', () => {
        console.log('suatu event telah terjadi');
});
```

- **eventEmitter.emit(eventName)**
  Method ini berfungsi untuk memicu suatu *event* yang akan dipancarkan. **Parameter:**
  - *eventName*, nama dari sebuah *event* yang akan dipancarkan.
  - *...args*, argumen tambahan yang akan diberikan pada *eventName*.

  Kembalian: *true* apabila *event* memiliki *listener*, false jika tidak.

  Method ini akan memanggil masing-masing *listener* yang sudah dicatat oleh *eventEmitter.on()* dalam *array of listeners* secara sinkronis, dimana beberapa *listener* tersebut mengacu pada *eventName* yang sama. Argumen yang diterima dari parameter akan diberikan pada masing-masing *listener*.

  Berikut merupakan contoh implementasi dari *method* ini:

```
const EventEmitter = require('events');

class MyEmmit extends EventEmitter {}

const myEmmit = new MyEmitter();

myEmmit.on('event', () => {
console.log('suatu event telah terjadi');
});
myEmitter.emit('event');
```

2.3.5 Stream

Kelas ini digunakan untuk menangani aliran data yang terjadi pada *Node.js*. Data yang ditangani dapat berjumlah banyak dan akan menghabiskan banyak memori apabila tidak ditangani dengan baik. Oleh karena itu, modul *stream* menyediakan fitur-fitur yang memudahkan penanganan aliran data.

Ada empat tipe dasar *stream* dalam *Node.js*:

- **Readable**
  Streams yang dapat membaca data dari sumber eksternal tertentu.
- **Writable**
  Streams yang dapat menulis data dan mengirimkannya ke sumber external tertentu.
- **Duplex**
  Streams yang dapat membaca dan menulis data sekaligus.
- **Transform**
  Duplex streams yang dapat memodifikasi atau mengubah data dimana data tersebut dapat dilihat langsung hasil perubahannya.

1. **Readable Stream**
   Merupakan abstraksi untuk sumber data yang digunakan. Berikut merupakan contoh dari *Readable Stream* pada *Node.js*:

- **HTTP responses** pada *client*
- **HTTP requests** pada *server*
- **fs read streams**

Seluruh *Readable streams* mengimplementasi *interface* yang didefinisikan oleh kelas *stream.Readable*.

- **stream.Readable**
  Events:
  - **'close'**
    Event ini dipancarkan saat suatu *stream* atau sumber lain telah ditutup. *Event* ini menandakan tidak akan ada *event* lagi yang akan dipancarkan, dan tidak ada komputasi lain yang akan dilakukan.

```
const readable = getReadableStreamSomehow();
readable.on('close', (chunk) => {
        console.log(‘Stream telah ditutup‘);
});
```

  - **'data'**
    Event ini akan dipancarkan setiap kali suatu *stream* melepas kepemilikan sebuah data kepada pemakai. Hal tersebut dapat terjadi setiap suatu *stream* berganti menjadi mode *flowing* dengan memanggil *readable.pipe()*, *readable.resume()*, atau dengan menghubungkan *listener callback* pada *'data' event*. Contoh implementasi:

```
const readable = getReadableStreamSomehow();
readable.on('data', (chunk) => {
        console.log(‘Menerima data sebesar ${chunk.length} bytes
});
```

  - **'end'**
    Event ini dipancarkan saat tidak ada lagi data yang akan digunakan dari *stream*. Contoh implementasi:

```
const readable = getReadableStreamSomehow();
readable.on('end', () => {
        console.log('Tidak akan ada data yang dikirimkan.');
});
```

  - **'error'**
    Event ini akan dipancarkan oleh *Readable* setiap saat. *Event* ini dapat terjadi apabila *stream* tidak dapat menyediakan data dikarenakan kesalahan internal, atau ketika implementasi suatu *stream* mencoba mengirimkan *chunk* dari data yang tidak sesuai. Contoh implementasi:

```
const readable = getReadableStreamSomehow();
readable.on('error', () => {
        console.error('Terjadi kesalahan');
});
```

  Method:
  - **readable.pipe(destination[, options])**
    Parameter:
    * **destination**
      tipe: *stream.Writable*
      Destinasi untuk menulis suatu data.

* **options**
  tipe: *Object*
  Bersifat opsional, dapat berupa objek sebagai berikut:
  · **end**
    tipe: *boolean*
    Mengakhiri *writer* yang melakukan proses menulis saat *reader* telah selesai. Nilai *default* parameter ini yaitu *true*.

Kembalian: referensi yang menuju *destination*.
Method ini akan menghubungkan *Writable stream* pada objek *readable*, sehingga dapat berubah menjadi mode *flowing* secara otomatis dan akan menaruh seluruh data pada *Writable* yang sudah terhubung. *Method* ini mengembalikan referensi yang menuju *destination*, sehingga *pipe streams* dapat saling menyambung dengan *pipe streams* lainnya.
Berikut merupakan contoh implementasi dari *method* ini:

```
const fs = require('fs');

const readable = getReadableSt
const writable = fs.createWri

//seluruh data dari objek rea
readable.pipe(writable);
```

2.4 Express.js

Express.js merupakan *framework* aplikasi web untuk *Node.js* [6]. *Express.js* menyediakan fitur-fitur untuk web dan aplikasi *mobile* agar dapat bertahan lama. Untuk dapat menggunakan *Express.js*, dapat dilakukan langkah sebagai berikut:

```
var express = require('express');
var app = express();
```

Dengan begitu, fitur-fitur yang terdapat pada *Express.js* dapat digunakan untuk pengembangan aplikasi tertentu.
subbab-subbab berikut akan menjelaskan kelas-kelas yang terdapat pada *Express.js*.

2.4.1 express()

Untuk membuat aplikasi *Express*, langkah yang dilakukan adalah sebagai berikut:

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

Method yang dimiliki oleh fungsi *express()* yaitu sebagai berikut:

- **express.Router([options])**
  Parameter:
  - **options** bersifat opsional dan akan menentukan sifat dari objek *router*. Parameter ini dapat berupa beberapa jenis seperti berikut:
    * *caseSensitive* memungkinkan *case-sensitive*. Dapat bernilai *true* atau *false*. Secara default akan bernilai *false*.
    * *strict* memungkinkan *strict routing*. Dapat bernilai *true* atau *false*. Apabila bernilai *false*, maka parameter *'/foo'* dan *'/foo/'* akan dianggap sama oleh *router*.

2.4.2 Application

Kelas ini akan menangani berbagai proses yang terjadi dalam aplikasi *Express* seperti melakukan *routing* terhadap *HTTP requests*, mengatur *middleware*, *rendering* sebuah *HTML views*, dan mendaftarkan *template engine* tertentu. Untuk dapat melakukan fungsi-fungsi tersebut dapat dilakukan langkah berikut:

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

Baris pertama dari potongan kode tersebut berarti variabel *express* memanggil modul *'express'* agar dapat mengakses fungsi-fungsi yang ada pada modul tersebut. Sedangkan baris kedua, Objek *app* memanggil fungsi *express()* yang telah didapatkan dari variabel *express*.

Kelas ini memiliki beberapa *method* sebagai berikut:

- **app.all(path, callback[, callback ...])**
  Parameter:
  - **path** suatu *path* yang akan ditangani oleh *middleware*. Dapat berupa *string*, *path pattern*, atau *array* dari kombinasi *string* dan *path pattern*.
  - **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut.

  Method ini dapat menangani seluruh *HTTP requests* seperti *GET, POST, PUT,* dan *DELETE*. Berikut merupakan contoh implementasi dari *method* ini:

```
app.all('/about', function(req, res, next){
        console.log('Mengakses bagian about ...');
        next(); //bagian ini akan menuju ke handler be
});
```

- **app.get(path, callback[, callback ...])**
  Parameter:
  - **path** suatu *path* yang akan ditangani oleh *middleware*. Dapat berupa *string*, *path pattern*, atau *array* dari kombinasi *string* dan *path pattern*.
  - **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut.

  Method ini akan mengarahkan *HTTP GET requests* pada *path* dengan fungsi *callback* tertentu. Berikut merupakan contoh implementasi dari *method* ini:

```
app.get('/', function(req, res){
        res.send('Mengirimkan GET request pada homepag
});
```

- **app.post(path, callback[, callback ...])**
  Parameter:
  - **path** suatu *path* yang akan ditangani oleh *middleware*. Dapat berupa *string*, *path pattern*, atau *array* dari kombinasi *string* dan *path pattern*.
  - **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut.

Method ini akan mengarahkan *HTTP POST requests* pada *path* dengan fungsi *callback* tertentu. Berikut merupakan contoh implementasi dari *method* ini:

```
app.post('/', function(req, res){
        res.send('Mengirimkan POST requests pada hom
});
```

- **app.route(path)**
  Parameter:
  - **path** suatu *path* yang akan ditangani oleh *middleware*. Dapat berupa *string*, *path pattern*, atau *array* dari kombinasi *string* dan *path pattern*.

  Method ini akan mengembalikan instansi dari satu *route*, yang kemudian dapat digunakan untuk menangani *HTTP request* dengan *middleware* tertentu. Berikut merupakan contoh implementasi dari *method* ini:

```
app.route('/buku').get(function(req, res){
        res.send('Mendapatkan suatu buku');
});
```

- **app.use([path,] callback[, callback...])**
  Parameter:
  - **path** suatu *path* yang akan ditangani oleh *middleware*. Dapat berupa *string*, *path pattern*, atau *array* dari kombinasi *string* dan *path pattern*.
  - **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut.

  Method ini akan menghubungkan *middleware* atau suatu fungsi tertentu dengan *path* yang sudah ditentukan. Dalam implementasi *method* ini, urutan penempatan pada baris kode sangat berpengaruh. Setelah *app.use()* dieksekusi, maka suatu *request* tidak akan mengeksekusi *middleware* yang ada dibawah baris kode *app.use()*. Berikut merupakan contoh implementasi dari *method* ini:

```
//request hanya akan sampai pada middleware ini
app.use(function(req, res){
        res.send('Hanya sampai sini saja');
});

//request tidak akan mengeksekusi baris ini
app.get('/', function(req, res){
        res.send('Hello World!');
});
```

- **app.listen(port, [hostname], [backlog], [callback])**
  Parameter:
  - **port** nomor yang akan dituju oleh server.
  - **hostname** *string* yang diberikan pada gawai tertentu agar dapat dikenali. Parameter ini bersifat opsional.
  - **backlog** nomor yang menentukan ukuran maksimal dalam antrian koneksi yang tertunda.Parameter ini bersifat opsional.

- **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut. Parameter ini bersifat opsional.

Method ini akan menghubungkan suatu koneksi pada *host* dan *port* yang sudah ditentukan. Berikut merupakan contoh implementasi dari *method* ini:

```
const express = require('express');
const app = express();
app.listen(3000);
```

2.4.3 Request

Sebuah objek dari kelas *Request* akan merepresentasikan *HTTP request* dan memiliki properti untuk *request query* seperti *body, HTTP headers* dan *parameters*.

Beberapa *method* yang ada pada kelas *Request* yaitu:

- **req.accepts(types)**
  Berfungsi untuk memeriksa apakah tipe konten tertentu dapat diterima atau tidak.
- **req.get(field)**
  Berfungsi untuk mengembalikan *HTTP request header* tertentu.
- **req.is(type)**
  Berfungsi untuk mengembalikan apakah benar atau salah *type* pada parameter sama dengan status *Content-Type* pada *HTTP header*.

2.4.4 Response

Sebuah objek dari kelas *Response* akan merepresentasikan respon *HTTP* yang dikirim oleh *Express* pada saat menerima *HTTP request*.

Beberapa *method* yang terdapat pada kelas *Response* yaitu:

- **res.send([body])**
  Parameter: *body* dapat berupa berbagai jenis objek seperti *Buffer*, *String*, dan *Array*.

  Method ini akan mengirimkan respon HTTP kepada *client* sesuai dengan parameter yang diterima. Berikut merupakan contoh implementasi dari *method* ini:

```
//parameter objek String
res.send('Hello World!');

//parameter objek Array
res.send([1,2,3]);

//parameter objek Buffer
res.send(new Buffer('<p>This is a Buffer</p>'));
```

- **res.end([data][, encoding])**
  Parameter:
  - **data** dapat berupa objek *String* atau *Buffer* yang akan dikirim saat mengakhiri proses respon.
  - **encoding** merubah suatu tipe data menjadi tipe data yang lain. Contoh beberapa tipe data yang tersedia yaitu *utf8, base64, ascii,* dan *hex*.

Method ini berfungsi untuk mengakhiri suatu proses respon. Apabila akan mengakhiri suatu respon tanpa memerlukan suatu data, maka dapat menggunakan *method* ini. Berikut merupakan contoh implementasi *method* ini:

```
app.get('/', function(req, res){
        res.end(); //apabila tidak memerlukan data.

        res.end('goodbye!'); // apabila memerlukan s
});
```

- **res.render(view[, locals][, callback])**
  Parameter:
  - **view** suatu *string* yang menunjukan *path* dari suatu *view file*.
  - **locals** suatu objek yang memiliki properti yang menunjukan variabel lokal dari *view*.
  - **callback** suatu fungsi *callback*.

  Method ini berfungsi untuk merubah *view file* dan mengirim *file* tersebut kepada *client*. Berikut merupakan contoh implementasi *method* ini:

```
app.get('/', function(req, res){
        res.render('about');  //akan merubah(render)
});
```

- **res.sendStatus(statusCode)**
  Parameter:
  - **statusCode** kode status *HTTP*.

  Method ini akan menetapkan kode status *HTTP* di parameter, dan akan mengirimkan bentuk *String* sebagai *body* dari respon. Berikut contoh implementasi *method* ini:

```
res.sendStatus(200); // akan mengirimkan 'OK' pada response body.
res.sendStatus(404); // akan mengirimkan 'Not Found' pada response b
res.sendStatus(500); // akan mengirimkan 'Internal Server Error' pad
```

- **res.status(code)**
  Parameter:
  - **code** kode status *HTTP*.

  Method ini akan menetapkan kode status *HTTP* untuk respon. Berikut merupakan contoh implementasi *method* ini:

```
res.status(403).end();
res.status(400).send('Bad Request');
```

- **res.json([body])**
  Parameter:
  - **body** dapat berupa tipe *JSON* apapun, seperti *array, String,* dan *Boolean*.

  Method ini berfungsi untuk mengirimkan respon *JSON*. Berikut merupakan contoh implementasi *method* ini:

```
res.json({ user: 'tobi', age: '27'});
```

2.4.5 Router

Objek dari kelas *Router* merupakan *instance* dari *middleware* dan *routes*. Setiap aplikasi *Express* memiliki *router* secara *built-in*.

Method yang dimiliki oleh *Router* yaitu sebagai berikut:

- **router.METHOD(path, [callback, ...] callback)**
  Parameter:
  - **path** suatu *path* yang akan ditangani oleh *middleware*. Dapat berupa *string*, *path pattern*, atau *array* dari kombinasi *string* dan *path pattern*.
  - **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut.

  Method ini menyediakan fungsionalitas *routing* dalam aplikasi *Express*, dimana *METHOD* merupakan salah satu *HTTP methods* seperti *GET, PUT,* dan *POST*, dalam huruf kecil. Dengan begitu, *method* ini dapat berupa *router.get(), router.post(),* dan *router.put()*.

  Berikut merupakan contoh implementasi dari *method* ini:

```
//menggunakan HTTP method GET
router.get('/', function(req, res){
        res.send('hello world');
});

//menggunakan HTTP method POST
router.post('/buku', function(req, res){
        res.send('mendapatkan buku');
});
```

2.5 Canvas API

Canvas API merupakan salah satu elemen *HTML5* yang digunakan untuk membuat gambar grafis dalam aplikasi web [7]. Teknologi ini memiliki fitur untuk membuat komposisi foto, membuat animasi, dan membuat *real-time video processiong* atau *rendering*.

Subbab-subbab berikut menjelaskan tentang beberapa *interface* dari *Canvas*.

2.5.1 HTMLCanvasElement

Interface ini menyediakan beberapa properti dan *method* untuk memanipulasi tata letak dan tampilan dari elemen *canvas*.

Beberapa properti yang dimiliki oleh *HTMLCanvasElement* yaitu :

- **HTMLCanvasElement.height**
  Merupakan bilangan integer positif yang merepresentasikan tinggi dari atribut *HTML* pada elemen *canvas* yang diinterpretasikan dalam piksel *CSS*. Apabila atribut tidak didefinisikan, atau atribut diisi dengan nilai negatif, maka akan digunakan nilai *default* yaitu 150.

- **HTMLCanvasElement.width**
  Merupakan bilangan integer positif yang merepresentasikan lebar dari atribut *HTML* pada elemen *canvas* yang diinterpretasikan dalam piksel *CSS*. Apabila atribut tidak didefinisikan, atau atribut diisi dengan nilai negatif, maka akan digunakan nilai *default* yaitu 300.

Beberapa *method* yang dimiliki oleh *HTLMCanvasElement* yaitu :

- **HTMLCanvasElement.getContext()**
  Method ini akan mengembalikan konteks *drawing* pada *canvas*, atau mengembalikan *null* apabila konteks *ID* tidak tersedia. Konteks *drawing* berfungsi untuk dapat menggambar pada *canvas*.
- **HTMLCanvasElemen.toBlob()**
  Method ini akan membuat objek *Blob* yang merepresentasikan gambar yang ada pada *canvas*.

2.5.2 CanvasRenderingContext2D

Interface ini digunakan untuk menggambar persegi panjang, teks, gambar, dan objek-objek lain kedalam elemen *canvas*. *CanvasRenderingContext2D* menyediakan konteks *2D rendering* untuk suatu elemen *<canvas>*. Untuk mendapatkan objek dari *interface* ini, harus memanggil *getContext()* didalam elemen *<canvas>*, dengan memberi "2d" sebagai argumen. Berikut contoh penggunaannya :

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

Rectangles

Ada tiga *method* yang dapat digunakan untuk menggambar bentuk persegi panjang:

- **CanvasRenderingContext2D.clearRect()**
  Berfungsi untuk mengatur semua piksel dalam persegi panjang yang didefinisikan dengan titik awal (x,y) dan ukuran (lebar, tinggi) menjadi hitam transparan, dan menghapus semua konten yang telah digambar sebelumnya.
- **CanvasRenderingContext2D.fillRect()**
  Berfungsi untuk menggambar persegi panjang dengan berisi warna tertentu, pada posisi (x,y) dengan ukuran yang ditentukan dari *width* dan *height*.
- **CanvasRenderingContext2D.strokeRect()**
  Berfungsi untuk menggambar hanya garis luar dari persegi panjang, pada posisi (x,y) dengan ukuran yang ditentukan dari *width* dan *height*.

Text

Berikut merupakan beberapa *method* yang digunakan untuk menggambar suatu teks:

- **CanvasRenderingContext2D.fillText()**
  Menggambar teks tertentu pada posisi (x,y).
- **CanvasRenderingContext2D.strokeText()**
  Menggambar garis luar dari suatu teks pada posisi (x,y).

Line Styles

Berikut merupakan beberapa properti yang digunakan untuk mengatur bagaimana sebuah garis akan digambar:

- **CanvasRenderingContext2D.lineWidth**
  Properti yang merepresentasikan tebal dari suatu garis. Nilai *default* dari properti ini yaitu 1.0.

- **CanvasRenderingContext2D.lineCap**
  Berfungsi untuk menentukan jenis ujung dari suatu garis. Nilai dari properti ini dapat berupa *round*, *square*, atau *butt*.

Text Styles

Berikut merupakan beberapa properti yang digunakan untuk mengatur bagaimana suatu teks digambar.

- **CanvasRenderingContext2D.font**
  Berfungsi untuk mengatur jenis *font* yang akan digunakan. Nilai *default* dari properti ini yaitu 10px *sans-serif*.
- **CanvasRenderingContext2D.textAlign**
  Berfungsi untuk mengatur penjajaran dari suatu teks.
- **CanvasRenderingContext2D.direction**
  Berfungsi untuk mengatur arah dari teks tertentu. Nilai dari properti ini dapat berupa kiri-ke-kanan, atau kanan-ke-kiri.

Fill and Stroke Styles

Fill style digunakan untuk memanipulasi warna dan *style* pada suatu bentuk, dan *stroke style* digunakan untuk memanipulasi garis luar pada suatu bentuk.

- **CanvasRenderingContext2D.fillStyle**
  Berfungsi untuk memberi warna yang akan digunakan didalam suatu bentuk tertentu.
- **CanvasRenderingContext2D.strokeStyle**
  Berfungsi untuk memberi warna yang akan digunakan pada garis luar suatu bentuk tertentu.

Gradients and Patterns

- **CanvasRenderingContext2D.createLinearGradient()**
  Berfungsi untuk membuat *linear gradient* sepanjang garis pada koordinat tertentu.
- **CanvasRenderingContext2D.createRadialGradient()**
  Berfungsi untuk membuat *radial gradient* pada koordinat tertentu.
- **CanvasRenderingContext2D.createPattern()**
  Berfungsi untuk membuat pola dengan menggunakan gambar yang sudah didefinisikan sebelumnya.

Paths

Beberapa *method* berikut ini dapat digunakan untuk memanipulasi *path* dari suatu objek:

- **CanvasRenderingContext2D.beginPath()**
  Berfungsi untuk memulai *path* baru dengan mengosongkan daftar dari *sub-paths*.
- **CanvasRenderingContext2D.closePath()**
  Berfungsi untuk memindahkan posisi ujung *pen* ke titik awal dari *sub-path* saat ini.
- **CanvasRenderingContext2D.moveTo()**
  Berfungsi untuk memindahkan posisi ujung *pen* saat ini ke koordinat (x,y).

- **CanvasRenderingContext2D.lineTo()**
  Berfungsi untuk menghubungkan titik terakhir pada *sub-path* ke koordinat (x,y).

- **CanvasRenderingContext2D.arc()**
  Berfungsi untuk menambahkan garis lengkung ke *path* yang berpusat pada posisi (x,y) dengan radius *r*, dimulai dari *startAngle* dan berakhir pada *endAngle* dengan arah gambar garis lengkung yang didefinisikan oleh *anticlockwise*.

- **CanvasRenderingContext2D.rect()**
  Berfungsi untuk membuat *path* persegi panjang pada posisi (x,y) dengan ukuran yang didefinisikan oleh *width* dan *height*.

DAFTAR REFERENSI

[1] Mozilla (2011) WebSockets. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Online; diakses 7-Oktober-2017].

[2] Rauch, G. (2011) Socket.io. https://socket.io/. [Online; diakses 7-Oktober-2017].

[3] Rauch, G. (2011) Socket.io Server API. https://socket.io/docs/server-api/. [Online; diakses 7-Oktober-2017].

[4] Rauch, G. (2011) Socket.io Client API. https://socket.io/docs/client-api/. [Online; diakses 7-Oktober-2017].

[5] Dahl, R. (2009) Node.js. https://nodejs.org/en/. [Online; diakses 7-Oktober-2017].

[6] Holowaychuk, T. (2010) Express.js. https://expressjs.com/. [Online; diakses 7-Oktober-2017].

[7] Apple (2004) Canvas API. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. [Online; diakses 7-Oktober-2017].

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

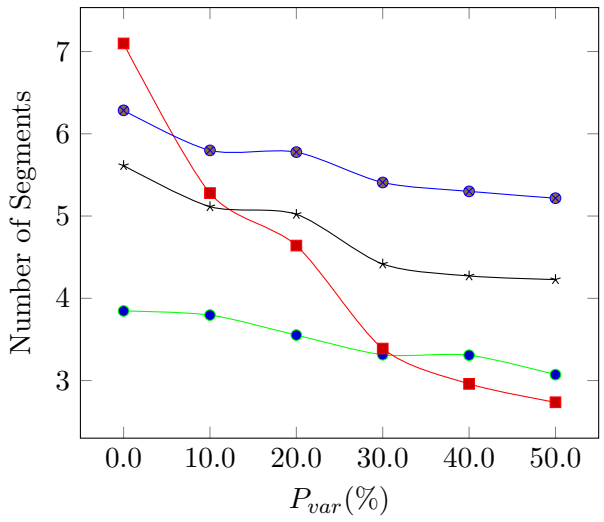
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

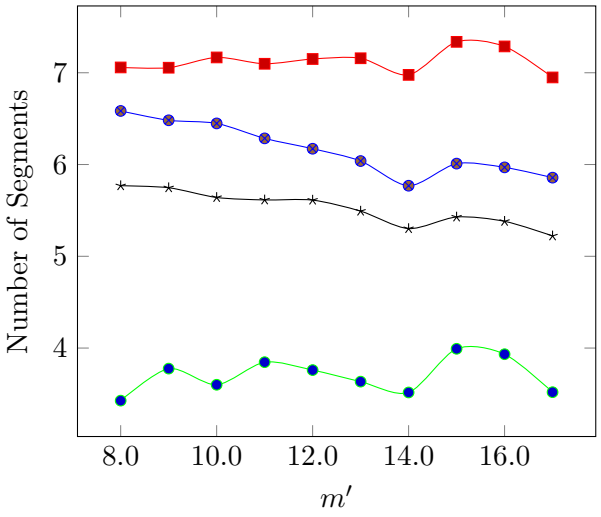
LAMPIRAN B

HASIL EKSPERIMEN

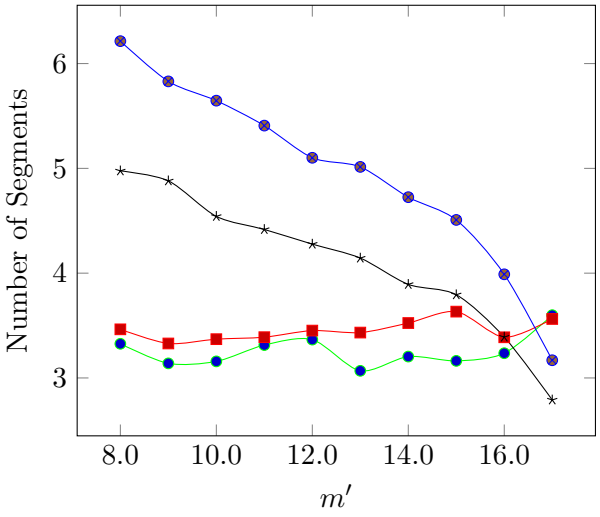
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



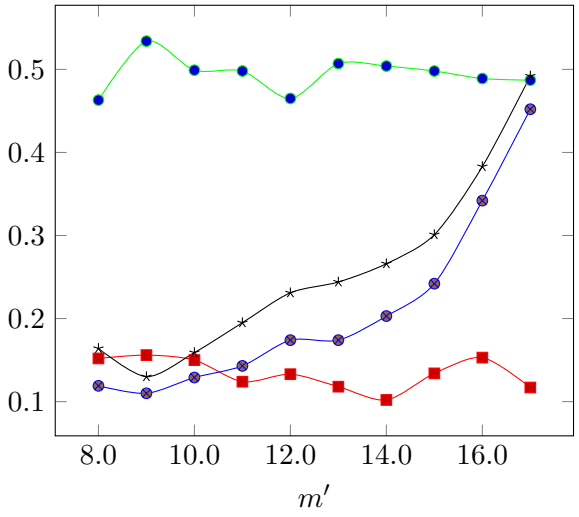
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4