

# 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 dijatuahkan 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

<b>KATA PENGANTAR</b>	<b>xv</b>
<b>DAFTAR ISI</b>	<b>xvii</b>
<b>DAFTAR GAMBAR</b>	<b>xix</b>
<b>DAFTAR TABEL</b>	<b>xxi</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	2
1.3 Tujuan . . . . .	2
1.4 Batasan Masalah . . . . .	2
1.5 Metodologi . . . . .	2
1.6 Sistematika Pembahasan . . . . .	3
<b>2 LANDASAN TEORI</b>	<b>5</b>
2.1 Node.js . . . . .	5
2.1.1 HTTP . . . . .	5
2.1.2 Path . . . . .	6
2.1.3 Module . . . . .	6
2.2 Express.js . . . . .	6
2.2.1 express() . . . . .	6
2.2.2 Application . . . . .	7
2.2.3 Response . . . . .	8
2.3 Socket.io . . . . .	9
2.3.1 Server API . . . . .	9
2.3.2 Client API . . . . .	11
2.4 Canvas API . . . . .	12
2.4.1 Animation . . . . .	13
2.4.2 canvasRenderingContext2D . . . . .	14
2.5 jQuery . . . . .	15
2.5.1 .submit(handler) . . . . .	15
2.5.2 .val() . . . . .	15
2.5.3 .html() . . . . .	16
2.5.4 .preventDefault() . . . . .	16
2.6 Content Template element . . . . .	16
<b>3 ANALISIS</b>	<b>17</b>
3.1 Analisis Aplikasi Sejenis . . . . .	17
3.2 Analisis Alur Permainan Finger For Life . . . . .	23
3.3 Analisis Pengembangan Web . . . . .	25
3.4 Analisis <i>Use Case</i> . . . . .	33

3.4.1	<i>Diagram Use Case</i>	33
3.4.2	<i>Skenario Use Case</i>	33
3.5	Analisis Aristektur Finger For Life	35
3.6	Analisis Socket.io	35
<b>4</b>	<b>PERANCANGAN</b>	<b>41</b>
4.1	Perancangan Sequence Diagram	41
4.1.1	<i>Sequence</i> Permintaan Bergabung	41
4.1.2	<i>Sequence</i> Memilih Karakter	43
4.1.3	<i>Sequence</i> Memulai Permainan	44
4.1.4	<i>Sequence</i> Mengakhiri Permainan	45
4.2	Perancangan Antarmuka	45
4.3	Perancangan Struktur Direktori	51
<b>5</b>	<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>63</b>
5.1	Implementasi	63
5.1.1	Lingkungan Implementasi	63
5.1.2	Hasil Implementasi	64
5.2	Pengujian	69
5.2.1	Pengujian Fungsional	69
5.2.2	Pengujian Eksperimental	73
<b>DAFTAR REFERENSI</b>		<b>75</b>
<b>A</b>	<b>KODE PROGRAM</b>	<b>77</b>
<b>B</b>	<b>HASIL EKSPERIMEN</b>	<b>79</b>

## DAFTAR GAMBAR

3.1 Halaman awal web pada <i>PC browser</i> . . . . .	17
3.2 Kode yang harus dimasukan oleh pemain pada <i>mobile browser</i> . . . . .	18
3.3 Halaman awal pada <i>mobile browser</i> . . . . .	18
3.4 Pemain diminta untuk memasukan kode yang sudah didapatkan pada <i>PC browser</i> . . . . .	19
3.5 Pemain diminta untuk memasukan kode yang sudah didapatkan pada <i>PC browser</i> . . . . .	19
3.6 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih. . . . .	20
3.7 Halaman pada <i>smartphone</i> yang berfungsi sebagai pengendali. . . . .	20
3.8 Halaman awal permainan The Neighborhood pada <i>PC</i> . . . . .	21
3.9 Halaman awal permainan The Neighborhood pada <i>smartphone</i> . . . . .	21
3.10 Halaman pada <i>PC</i> dimana permainan sedang berlangsung. . . . .	21
3.11 Halaman pada <i>smartphone</i> dimana permainan sedang berlangsung. . . . .	22
3.12 Halaman pada <i>PC</i> apabila permainan sudah dimenangkan. . . . .	22
3.13 Halaman pada <i>smartphone</i> apabila permainan sudah dimenangkan. . . . .	22
3.14 Halaman pada <i>PC</i> yang menunjukan pemutusan koneksi. . . . .	23
3.15 Efek <i>destination-over</i> . . . . .	30
3.16 Diagram <i>use case</i> pemain . . . . .	33
3.17 Arsitektur Finger For Life . . . . .	35
3.18 Arsitektur interaksi <i>client</i> dan <i>server</i> . . . . .	36
4.1 Proses melakukan koneksi ke <i>socket.io</i> dan bergabung kedalam <i>room</i> . . . . .	41
4.2 Proses memilih karakter. . . . .	43
4.3 Proses memulai permainan. . . . .	44
4.4 Menampilkan para pemain yang telah selesai bermain. . . . .	45
4.5 Halaman pada <i>PC</i> yang menunjukan tampilan awal saat <i>client</i> mengakses alamat web. . . . .	46
4.6 Halaman pada <i>mobile</i> yang tampilan awal saat <i>client</i> mengakses alamat web. . . . .	46
4.7 Halaman pada <i>PC</i> yang menampilkan langkah untuk bergabung dalam permainan	47
4.8 Halaman pada <i>smartphone</i> yang menampilkan kolom untuk mengisi kode. . . . .	47
4.9 Halaman pada <i>PC</i> yang menampilkan karakter yang telah ditetapkan oleh pemain.	48
4.10 Halaman pada <i>smartphone</i> yang menampilkan daftar karakter yang dapat dipilih.	48
4.11 Halaman pada <i>PC</i> yang menampilkan lintasan lari dan karakter untuk dimainkan.	49
4.12 Halaman pada <i>smartphone</i> yang menampilkan telapak kaki yang berfungsi sebagai pengendali. . . . .	50
4.13 Halaman pada <i>PC</i> yang menampilkan pemenang permainan. . . . .	50
4.14 Halaman pada <i>smartphone</i> yang menampilkan teks bahwa permainan telah selesai.	51
4.15 Direktori seluruh folder dan <i>file</i> . . . . .	51
4.16 Isi folder <i>bin</i> . . . . .	52
4.17 Isi dari folder <i>public</i> . . . . .	54
4.18 Isi dari folder <i>routes</i> . . . . .	60
4.19 Isi dari folder <i>views</i> . . . . .	60
5.1 Halaman <i>Home</i> . . . . .	65
5.2 Halaman <i>Home</i> . . . . .	65

5.3 Halaman <i>Home</i> . . . . .	66
5.4 Halaman <i>Home</i> . . . . .	66
5.5 Halaman <i>Home</i> . . . . .	67
5.6 Halaman <i>Home</i> . . . . .	67
5.7 Halaman <i>Home</i> . . . . .	68
5.8 Halaman <i>Home</i> . . . . .	68
5.9 Halaman <i>Home</i> . . . . .	69
5.10 Halaman <i>Home</i> . . . . .	69
B.1 Hasil 1 . . . . .	79
B.2 Hasil 2 . . . . .	79
B.3 Hasil 3 . . . . .	79
B.4 Hasil 4 . . . . .	79

## **DAFTAR TABEL**



# BAB 1

## PENDAHULUAN

### 1.1 Latar Belakang

Socket.io adalah teknologi yang memungkinkan untuk melakukan komunikasi secara *realtime*, dan dua arah antara *client* dan *server* [1]. Socket.io memiliki dua bagian: *client-side library* yang berjalan didalam *web browser*, dan *server-side library* yang berjalan pada bagian *server*. 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 teknologi Socket.io 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* [2]. *Canvas API* dapat juga digunakan untuk membuat komposisi foto dan membuat animasi. 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 teknologi Socket.io adalah Node.js. Teknologi ini merupakan sebuah *platform* yang didesain untuk mengembangkan aplikasi berbasis web pada bagian *web server* [3]. 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. Fitur-fitur yang dimiliki oleh Node.js akan sangat membantu untuk membangun aplikasi permainan yang membutuhkan koneksi *real-time*.

Salah satu teknologi yang akan membantu dalam memanfaatkan Node.js adalah Express.js [4]. Teknologi ini menyediakan kumpulan fitur untuk mengatur penyimpanan data lokal dalam membangun aplikasi web maupun *mobile*. Express.js hanya dapat digunakan untuk membangun aplikasi apabila aplikasi tersebut berjalan berdasarkan *Node.js*. Oleh karena itu, fitur-fitur yang dimiliki oleh Express.js akan membantu dalam pembangunan aplikasi berbasis Node.js

Pada skripsi ini, akan dibuat sebuah aplikasi permainan yang memanfaatkan Socket.io. Selain itu, aplikasi yang dibuat akan memanfaatkan *personal computer (PC)* dan *smartphone* untuk pengembangan aplikasinya. Para pemain akan mengkoneksikan *smartphone* pada suatu *PC* yang akan berfungsi sebagai *console*, dan *smartphone* tersebut akan berfungsi sebagai *controller* untuk memainkan permainannya. Oleh karena itu, Socket.io 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* harus mengakses melalui *web browser*.

## 1.2 Rumusan Masalah

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

## 1.3 Tujuan

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

## 1.4 Batasan Masalah

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

1. Studi literatur mengenai :
  - *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.
  - *jQuery* yang akan digunakan dalam pengaturan elemen HTML.
  - *The Content Template element* yang akan digunakan untuk menampilkan halaman-halaman HTML.
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 *Socket.io*, *Node.js*, *Express.js*, *Canvas API*, *jQuery*, dan *The Content Template element*.

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 Node.js, Express.js, Socket.io, Canvas API, jQuery, dan The Content Template element.

### 2.1 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* adalah sebagai berikut:

#### 2.1.1 HTTP

*Interfaces HTTP* pada *Node.js* digunakan untuk menangani *request* dari protokol *HTTP* yang secara *native* sulit untuk digunakan [3]. *Interface* ini akan menangani protokol *HTTP* dengan tidak melakukan *buffer* pada seluruh *request* atau *responses*.

Subbab berikut akan menjelaskan salah satu kelas yang ada pada *interface HTTP*.

##### 1. **http.Server**

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

- **server.listen()**

Memulai *server HTTP* melakukan proses *listening* untuk suatu koneksi.

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

- **http.createServer([options][,requestListener])**

**Parameter:**

- **options** objek-objek sebagai berikut:

- \* **IncomingMessage** menentukan objek dari kelas *IncomingMessage* yang akan digunakan.

- \* **ServerResponse** menentukan objek dari kelas *ServerResponse* yang akan digunakan.

- **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*.

### 2.1.2 Path

Modul *Path* menyediakan fungsi untuk mengatur akses suatu *file* dan direktori [3]. Modul tersebut dapat diakses dengan cara sebagai berikut:

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

Salah satu method yang dimiliki oleh modul *Path* adalah :

- **path.join([...paths])**

**parameter:**

- **...paths**

tipe: **String**

Urutan suatu lokasi *file* yang akan digunakan.

**kembalian:** String

*Method* ini akan menggabungkan seluruh bagian-bagian *path* dengan menormalisasinya dan mengembalikan bentuk *path* yang menyeluruh.

### 2.1.3 Module

Pada aplikasi berbasis *Node.js*, setiap *file* yang terdapat dalam pembangunan aplikasi dianggap sebagai modul-modul yang terpisah satu sama lain [3]. Variabel dan fungsi yang terdapat pada satu *file*, atau modul, hanya dapat digunakan pada satu lingkup modul tersebut. Suatu modul tidak dapat menggunakan variabel atau fungsi yang terdapat pada modul lainnya. Oleh karena itu, apabila variabel dan fungsi yang terdapat pada satu modul dapat digunakan oleh modul yang lain, diperlukan cara tertentu. Cara tersebut dapat dilakukan seperti berikut:

```
module.exports = functions || object
```

*module.exports* merupakan objek yang dibentuk oleh sistem *Module*. *Functions* dan *object* merupakan fungsi dan objek yang merupakan elemen-elemen dari modul tertentu, yang dapat dirubah menjadi global agar dapat diakses oleh modul lain. Dengan menggunakan cara ini, suatu modul dapat berubah menjadi global dan dapat diakses oleh modul-modul lain.

## 2.2 Express.js

*Express.js* merupakan *framework* aplikasi web untuk *Node.js* [4]. *Express.js* menyediakan fitur-fitur untuk web dan aplikasi *mobile* agar dapat bertahan lama. *Framework* ini digunakan untuk mengatur struktur direktori dalam pengembangan aplikasi permainan berbasis web. 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*.

### 2.2.1 express()

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

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

Beberapa *method* yang dimiliki oleh fungsi *express()* adalah sebagai berikut:

- **express.static(root, [options])**

**Parameter:**

- **root**

tipe: **String**

Menentukan direktori *root* yang akan digunakan untuk menyediakan *static file*.

- **options** Objek-objek seperti berikut:

- \* **dotfiles** menentukan bagaimana mengatasi suatu *dotfiles* (suatu *file* atau direktori yang dimulai dengan tanda ".").

*Method* ini akan menyediakan cara agar dapat menggunakan *static file* yang ada.

- **express.Router([options])**

**Parameter:**

- **options** merupakan parameter pilihan yang akan menentukan perilaku dari *server*. Parameter dapat berupa objek sebagai berikut:

- \* **caseSensitive** membedakan huruf besar dan huruf kecil.

*Method* ini akan membuat objek *router* yang dapat digunakan dengan menambahkan *middleware* dan *method HTTP* seperti *get*, *post*, dan *put*.

## 2.2.2 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 [6]. 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.set(name, value)**

**Parameter:**

- **name** nama tertentu yang dapat digunakan untuk menentukan perilaku dari suatu *server*.

- **value** nilai yang akan ditetapkan pada parameter *name*.

**Kembalian:** -

Method ini akan menetapkan suatu *value* tertentu pada parameter *name*.

- **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()*.

- **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.

### 2.2.3 Response

Sebuah objek dari kelas *Response* akan merepresentasikan respon *HTTP* yang dikirim oleh *Express* pada saat menerima *HTTP request* [6].

Salah satu properti yang dimiliki oleh kelas ini adalah:

- **res.locals**

Objek yang berisi variabel lokal milik *response* yang berada dalam lingkup suatu *request* tertentu. Objek ini hanya tersedia selama waktu *request* atau *response* tertentu.

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

- **res.render(view[, locals][, callback])**

Parameter:

- **view** suatu *string* yang menunjukkan *path* dari suatu *view file*.
- **locals** suatu objek yang memiliki properti yang menunjukkan variabel lokal dari *view*.
- **callback** suatu fungsi *callback*.

*Method* ini berfungsi untuk merubah *view file* dan mengirim *file* tersebut kepada *client*.

- **res.status(code)**

Parameter:

- **code** kode status *HTTP*.

*Method* ini akan menetapkan kode status *HTTP* untuk respon.

## 2.3 Socket.io

*Socket.io* merupakan salah satu teknologi yang memanfaatkan protokol *WebSockets* [1]. Teknologi ini memungkinkan sebuah aplikasi untuk melakukan komunikasi dua arah secara *real-time*. *Socket.io* dapat dijalankan di setiap *platform*, *browser*, dan gawai.

Sebelum dapat menggunakan *socket.io*, *Node.js* harus sudah terinstall pada sistem komputer. Apabila hal tersebut sudah dilakukan, maka *socket.io* dapat diinstall dengan menggunakan *command line tools* atau sejenisnya dengan melakukan langkah seperti berikut:

```
npm install socket.io
```

Dengan begitu, aplikasi yang dibuat sudah dapat mengakses fitur-fitur yang dimiliki oleh *socket.io*.

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

### 2.3.1 Server API

Kelas-kelas yang ada pada *Server API* digunakan untuk menangani proses yang terjadi dalam *server*[7]. 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 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.

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

- **server.listen(port[, options])**

**Parameter:**

- **port**

tipe: **Number**

Nomor yang akan digunakan untuk melakukan koneksi kepada *server*

- **options**

tipe: **Object**

Parameter ini dapat berupa berbagai jenis objek.

*Method* ini akan melakukan koneksi kepada *server* dengan menggunakan *port* yang terdapat pada parameter.

## 2. Namespace

Kelas ini merepresentasikan kumpulan *sockets* yang terhubung dalam lingkup yang diidentifikasi oleh nama *path*. *Client* akan selalu terhubung ke / (*namespace* utama), kemudian dapat terhubung ke *namespace* lain saat berada dalam koneksi yang sama.

Beberapa *method* yang dimiliki oleh kelas ini yaitu:

- **namespace.emit(eventName[, ...args])**

Berfungsi untuk memancarkan suatu *event* pada seluruh *clients* yang terhubung.

**Parameter:**

- **eventName**  
tipe: **String**  
Nama dari *event*.
- **args**  
Argumen tambahan.

Contoh implementasi:

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

```
// akan memancarkan event pada namespace utama (/)
io.emit('an event sent to all connected clients');
```

- **namespace.to(room)**

Berfungsi untuk memancarkan *event* kepada *client* yang sudah bergabung dalam *room* tertentu.

**Parameter:**

- **room**  
tipe: **String**  
Nama dari *room*.

**Kembalian:** *namespace*.

Contoh implementasi:

```
const io = require('socket.io')();
const adminNamespace = io.of('/admin');
```

```
adminNamespace.to('level1').emit('an event', { some: 'data' });
```

## 3. Socket

Kelas ini merupakan kelas yang mendasar untuk berinteraksi dengan *browser* milik *clients*. *Socket* merupakan milik *namespace* tertentu dan menggunakan kelas *Client* untuk berkomunikasi. Dalam setiap *namespace*, dapat ditentukan suatu *room* yang dimana sebuah *socket* dapat bergabung atau keluar. Kelas ini pun merupakan turunan dari *EventEmitter* milik *Node.js*. Kelas ini melakukan *override* pada *method* *emit* milik *EventEmitter*, dan tidak memodifikasi *method* lain.

Beberapa properti yang dimiliki oleh kelas ini yaitu sebagai berikut:

- **socket.id**

Tanda pengenal unik untuk sesi saat ini, yang didapatkan dari kelas *Client*.

- **socket.rooms**

Objek yang menandakan *room* dari *client* saat ini.

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

- **socket.join(room[, callback])**

Berfungsi untuk menambah *client* ke *room*.

**Parameter:**

- **room**

tipe: **String**

Nama *room*.

- **callback**

tipe: **Function**

Fungsi *callback*.

**Kembalian:** *Socket*.

### 2.3.2 Client API

*Client API* digunakan untuk menangani proses pengaturan koneksi yang terjadi pada bagian *client*[8]. Agar dapat menggunakan *API* yang tersedia, *client* harus menambahkan *url* pada *syntax script* didalam *html* yang bersangkutan. Hal tersebut dapat dilakukan seperti berikut:

```
<script src="/socket.io/socket.io.js"></script>
```

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

#### 1. IO

Untuk dapat menggunakan fungsi yang ada pada *IO*, dapat dilakukan langkah seperti berikut:

```
// berfungsi untuk melayani file client
<script src="/socket.io/socket.io.js"></script>
```

*Method* yang dimiliki oleh kelas ini yaitu sebagai berikut:

- **io([url][, options])**

Berfungsi untuk membuat objek baru dari kelas *Manager* dengan *url*, dan akan menggunakan objek kelas *Manager* yang sudah ada untuk pemanggilan selanjutnya, apabila *multiplex* pada parameter *option* bernilai *true*. Objek *Socket* akan dikembalikan untuk *namespace* yang sudah ditentukan oleh nama *path* pada *URL*, dengan nilai *default* /.

**Parameter:**

- **url**

tipe: **String**

Nama *URL*.

- **options**

tipe: **Object**

Parameter ini dapat berupa beberapa jenis objek, seperti milik kelas *Manager*

**Kembalian:** *Socket*

#### 2. Socket

Salah satu properti yang dimiliki oleh kelas ini yaitu:

- **socket.id**

*Identifier* unik yang dimiliki oleh satu *socket* untuk satu sesi. Properti ini akan mempunyai nilai segera setelah koneksi terhubung, dan akan diperbarui setelah melakukan koneksi ulang.

Beberapa *method* yang dimiliki oleh kelas ini yaitu:

- **socket.on(eventName, callback)**

Berfungsi untuk menyediakan fungsi yang menangani *event* yang dipancarkan.

**Parameter:**

- **eventName**, nama *event* yang dipancarkan.
- **callback**, fungsi yang akan menangani *event* yang dipancarkan.

- **socket.emit(eventName[, ..args][, ack])**

Berfungsi untuk memancarkan *event* kepada *socket* yang ditandai dengan nama dari *event* tersebut.

**Parameter:**

- **eventName**  
tipe: **String**  
Nama *event*.
- **args**  
Argumen tambahan (opsional).
- **ack**  
tipe: **Function**  
Fungsi tambahan (opsional).

Contoh implementasi:

```
socket .emit( ' ferret ' , ' tobi ' , ( data ) => {
    console .log( data );
});
```

Beberapa *events* yang ada pada kelas ini yaitu sebagai berikut:

- **connect**

Akan dipancarkan apabila berhasil melakukan koneksi dan setelah melakukan koneksi ulang.

- **disconnect**

Akan dipancarkan apabila *server* memutus koneksi atau *client* memutus koneksi.

## 2.4 Canvas API

Canvas API merupakan salah satu elemen *HTML5* yang digunakan untuk membuat gambar grafis dalam aplikasi web [2]. Teknologi ini memiliki fitur untuk membuat komposisi foto dan membuat animasi. Untuk dapat menggunakan fitur-fitur yang ada pada *Canvas API*, langkah yang harus dilakukan adalah sebagai berikut:

1. Menambahkan tag *<canvas>* pada file *HTML*, dan menambahkan *id* yang akan digunakan pada file *JavaScript*.

```
<canvas id="canvas"></canvas>
```

2. Membuat variabel untuk mendapatkan konteks *rendering* dan fungsi-fungsi menggambar agar dapat menampilkan sesuatu pada *<canvas>*.

```
// Variabel yang akan menampilkan sesuatu
// pada <canvas> dengan id='canvas'
var canvas = document.getElementById('canvas');
```

```
// Variabel yang akan mendapatkan fungsi-fungsi menggambar
var ctx = canvas.getContext('2d');
```

Subbab-subbab berikut akan menjelaskan tentang beberapa elemen yang tersedia di *Canvas API*.

### 2.4.1 Animation

Dengan menggunakan *JavaScript* untuk mengontrol elemen `<canvas>`, hal tersebut akan sangat membantu dalam membuat animasi interaktif. Subbab ini akan membahas tentang pembuatan animasi didalam *Canvas API*.

#### Langkah Dasar Animasi

Ada beberapa langkah yang dibutuhkan untuk menggambar suatu *frame*.

1. *Clear the canvas*

Gambar atau bentuk yang berada dalam *canvas* pada *state* sebelumnya harus dihapus terlebih dahulu. Cara yang paling mudah untuk melakukan hal tersebut adalah menggunakan *method* `clearRect()`.

2. *Save the canvas state*

Apabila akan dilakukan perubahan pada gaya, transformasi, atau hal lainnya yang mengakibatkan kondisi *canvas* saat ini, maka kondisi *canvas* yang asli harus disimpan terlebih dahulu. Hal tersebut dilakukan agar kondisi *canvas* yang asli digunakan setiap suatu *frame* digambar.

3. *Draw animated shapes*

Langkah ini melakukan proses *rendering* pada *frame* yang telah digambar.

4. *Restore the canvas state*

Apabila kondisi *canvas* telah disimpan sebelumnya, maka kondisi tersebut harus dikembalikan lagi sebelum menggambar *frame* baru.

Untuk dapat melihat suatu gambar atau bentuk yang bergerak didalam *canvas*, diperlukan cara menjalankan fungsi menggambar selama periode waktu tertentu. Berikut merupakan beberapa cara yang dapat dilakukan:

- `setInterval(func, delay[, param1, param2, ...])`

**Parameter:**

- **func**

Suatu fungsi yang akan dieksekusi setiap *delay milliseconds*.

- **delay**

Waktu dalam *milliseconds*. *Timer* akan melakukan *delay* diantara waktu selama menjalankan fungsi tertentu.

- **param1,...,paramN**

Parameter tambahan untuk fungsi apabila waktu *timer* telah habis.

**Kembalian:**

**intervalID** identifikasi unik milik suatu interval.

*Method* ini akan memanggil suatu fungsi atau mengeksekusi kode tertentu selama rentang waktu yang telah ditentukan. Pemanggilan fungsi atau eksekusi kode dilakukan dengan penundaan selama waktu *delay* tertentu.

- `clearInterval(intervalID)`

**Parameter:**

- **intervalID** identifikasi milik suatu interval yang akan diberhentikan.

*Method* ini akan memberhentikan suatu fungsi yang berjalan berulang-ulang selama rentang waktu tertentu, yang dipanggil oleh *method setInterval()*.

- **requestAnimationFrame(callback)**

**Parameter:**

- **callback**

Parameter fungsi yang akan dipanggil saat harus memperbarui animasi dan menggambar *frame* selanjutnya.

*Method* ini akan memberitahu *browser* bahwa akan dilakukan suatu animasi, dan melakukan *request* kepada *browser* untuk memanggil fungsi tertentu untuk memperbarui animasi sebelum melakukan gambar ulang selanjutnya.

#### 2.4.2 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');
```

Salah satu properti yang dimiliki oleh *interface* ini adalah:

- **CanvasRenderingContext2D.globalCompositeOperation**

Properti ini akan menetapkan tipe operasi komposisi yang akan digunakan saat menggambar suatu bentuk baru kedalam *canvas*. Tipe tersebut berupa *String* yang akan menentukan komposisi atau *blending mode* yang akan digunakan.

Berikut contoh tipe yang dapat digunakan:

- **destination-over** Bentuk yang baru akan digambar pada posisi dibelakang konten *canvas* yang telah ada sebelumnya.

Beberapa *method* yang dimiliki oleh *interface* ini adalah sebagai berikut:

- **CanvasRenderingContext2D.clearRect(x, y, width, height)**

*Method* ini akan menghapus gambar sebelumnya dengan membentuk suatu persegi. *Method* ini akan menggambar koordinat titik awal (*x, y*) dengan lebar dan tinggi yang sudah ditentukan oleh *width* dan *height*.

**Parameter:**

- **x**

Koordinat x yang menandakan titik awal persegi.

- **y**

Koordinat y yang menandakan titik awal persegi.

- **width**

Lebar persegi.

- **height**

tinggi persegi.

- **CanvasRenderingContext2D.drawImage(image, dx, dy)**

**Parameter:**

- **image** elemen yang akan digambar kedalam *context* tertentu.
- **dx** koordinat *x* pada *canvas* untuk menempatkan *image* di pojok kiri atas.
- **dy** koordinat *y* pada *canvas* untuk menempatkan *image* di pojok kiri atas.

*Method* ini akan menyediakan cara untuk menggambar suatu elemen *image* pada *canvas*.

- **CanvasRenderingContext2D.save()**

*Method* ini akan menyimpan seluruh *state* dari *canvas* dengan menaruh *state* tersebut kedalam suatu *stack* yang sudah diatur didalam elemen <*canvas*>.

- **CanvasRenderingContext2D.restore()**

*Method* ini akan mengembalikan *state* yang baru saja disimpan dengan mengeluarkannya dari tumpukan paling atas suatu *stack*. Apabila *stack* tersebut kosong, maka *method* ini tidak melakukan apapun.

## 2.5 jQuery

jQuery merupakan pustaka JavaScript yang menyediakan fitur-fitur untuk mengatur berbagai elemen HTML [9]. Pustaka ini memiliki fitur-fitur seperti memanipulasi berkas HTML, menangani suatu *event*, dan mengatur jalannya animasi. jQuery dapat menyediakan fitur-fitur untuk menangani berbagai hal tersebut yang berjalan di berbagai *browser* yang berbeda.

Subbab-subbab berikut akan menjelaskan beberapa *method* yang dimiliki oleh jQuery.

### 2.5.1 .submit(handler)

*Method* ini akan menyambungkan *handler* suatu *event* dengan "submit" yang merupakan *event* dari JavaScript, atau memancarkan *event* tersebut kepada elemen tertentu [10].

**Parameter:**

- **handler**, Fungsi yang akan dieksekusi setiap suatu *event* dipancarkan.

Berikut merupakan contoh penggunaan dari *method* ini:

```
// HTML
<form id="target">
<input type="text" value="Hello there">
<input type="submit" value="Go">
</form>

// JQUERY
$( "#target" ).submit( function( event ) {
  alert( "Handler for .submit() called ." );
  event.preventDefault();
});
```

### 2.5.2 .val()

*Method* ini akan mendapatkan nilai dari elemen-elemen *form* seperti *input*, *select*, dan *textarea* [10].

**Kembalian:** *String, Number, Array*.

### 2.5.3 .html()

*Method* ini akan mendapatkan konten HTML dari elemen pertama yang ada didalam kumpulan elemen-elemen yang sesuai [10].

**Kembalian:** *String*

### 2.5.4 .preventDefault()

Apabila *method* ini dieksekusi, maka aksi *default* dari suatu *event* tidak akan dieksekusi [10].

## 2.6 Content Template element

HTML Content Template (<template>) element adalah suatu mekanisme untuk menyimpan konten milik *client* agar konten tersebut tidak dimuat pada saat memuat halaman, dimana konten tersebut dapat dipakai pada saat *runtime* dengan menggunakan JavaScript [11].

Berikut merupakan contoh penggunaan dari elemen <template>:

```
<html>
<head>
<title>Home</title>
</head>
<body>
<p>Hello World!</p>

<template id="stage">

</template>

<template id="home_page">
<p>Hello from template element!</p>
</template>

</body>
</html>
```

Apabila halaman HTML ini dimuat, maka yang akan ditunjukan dihalaman adalah teks "Hello World!". Seluruh elemen yang ada didalam elemen <template> tidak akan dimuat dihalaman. Agar seluruh isi dari elemen <template> dimuat, maka diperlukan JavaScript. Berikut merupakan contoh kode JavaScript untuk memuat elemen <template>.

```
<script>
// variable yang menyimpan elemen dari <template> stage
var bg = $("#stage");

// variable ini menyimpan seluruh isi dari template home_page
var home = $("#home_page").html();

// variable bg akan memuat template dari home_page
bg.html(home);

</script>
```

## BAB 3

### ANALISIS

Pada bab ini akan dijelaskan mengenai analisis aplikasi sejenis yang menggunakan *smartphone* sebagai pengendali permainan berbasis web, analisis *sequence diagram*.

#### 3.1 Analisis Aplikasi Sejenis

Salah satu aplikasi sejenis permainan berbasis web dengan memanfaatkan *smartphone* sebagai pengendali yaitu AirConsole. Aplikasi tersebut memanfaatkan teknologi *browser*, *smartphone*, *PC*, dan juga jaringan internet untuk dapat menggunakananya. Aplikasi ini dikembangkan oleh N-Dream AG.

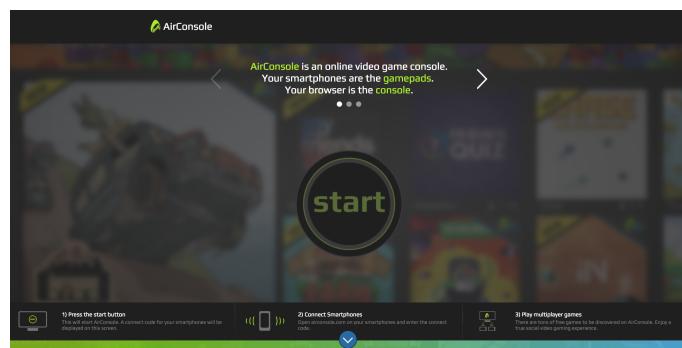
##### Analisis AirConsole

AirConsole merupakan permainan berbasis web dimana *browser* pada *mobile device* dapat melakukan koneksi ke *browser* pada *PC*. Pada aplikasi ini, terdapat berbagai macam permainan yang dapat dipilih oleh pemain. Untuk dapat memainkan aplikasi tersebut, pemain harus membuka alamat web *airconsole.com* pada *PC browser* dan juga pada *smartphone browser*.

Analisis dilakukan dengan cara berikut:

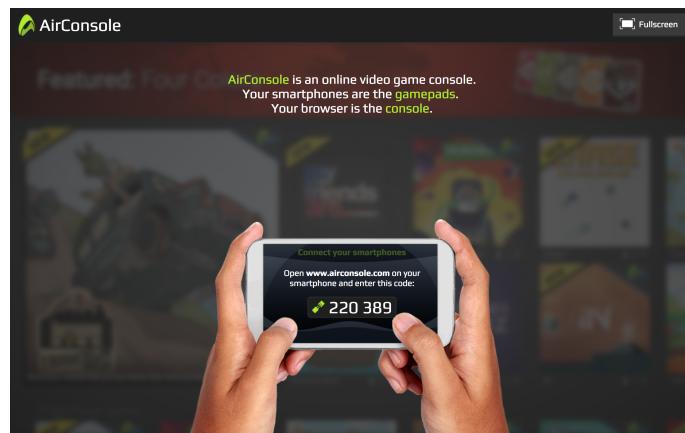
1. Memainkan permainan dari awal hingga akhir.
2. Keluar dari *browser* pada *PC* pada saat permainan berlangsung.
3. Keluar dari *browser* pada *smartphone* pada saat permainan berlangsung.

Pada halaman awal web di *PC*, pemain diminta untuk menekan tombol *start* yang ada pada gambar berikut:



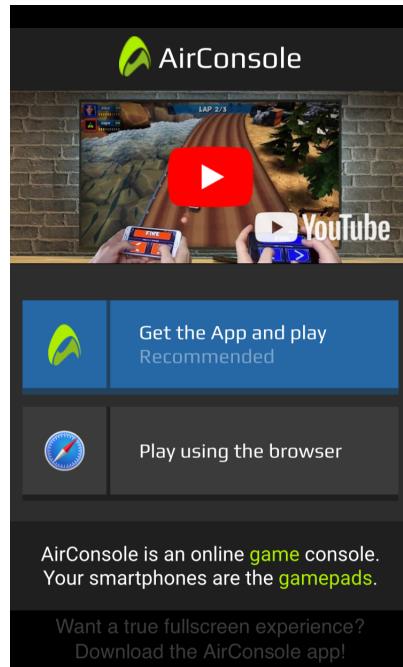
Gambar 3.1: Halaman awal web pada *PC browser*.

Setelah tombol *start* ditekan, maka akan muncul halaman berikutnya yang menunjukkan kode yang harus dimasukan oleh pemain pada *mobile browser*.



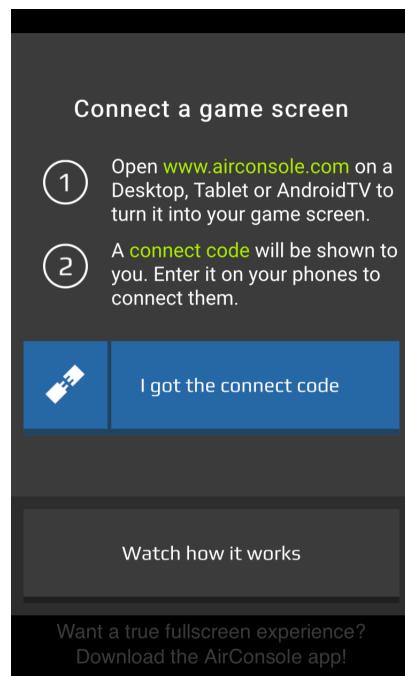
Gambar 3.2: Kode yang harus dimasukan oleh pemain pada *mobile browser*.

Pemain harus mengakses alamat web yang sama pada *mobile browser*. Pada halaman awal, pemain akan diminta untuk memilih apakah akan bermain dengan menggunakan aplikasi, atau bermain dengan menggunakan *browser*.



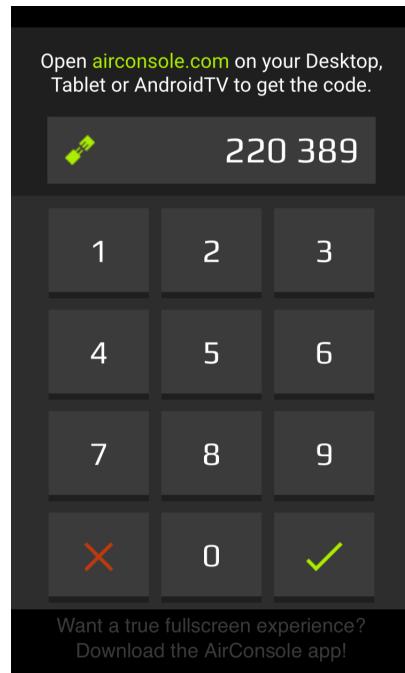
Gambar 3.3: Halaman awal pada *mobile browser*.

Dalam analisis ini, penulis memilih untuk bermain menggunakan *browser*. Setelah itu, pemain diminta untuk menekan tombol '*i got the connect code*' untuk memasukan kode yang sudah didapatkan pada *PC browser*.



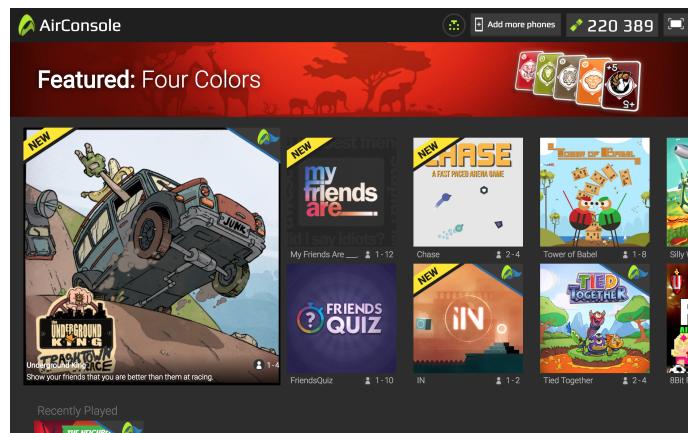
Gambar 3.4: Pemain diminta untuk memasukan kode yang sudah didapatkan pada *PC browser*.

Setelah menekan tombol tersebut, pemain dapat mulai memasukan kode yang sudah didapatkan. Kode ini bertujuan untuk proses otentikasi, sehingga para pemain yang dapat bermain dalam satu sesi yang sama, hanya para pemain yang mengetahui kode tersebut.

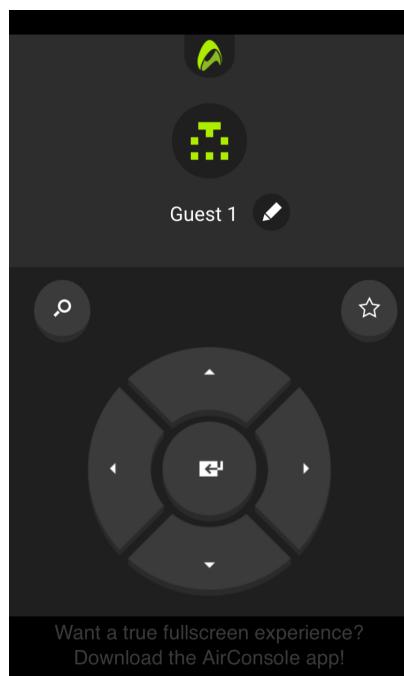


Gambar 3.5: Pemain diminta untuk memasukan kode yang sudah didapatkan pada *PC browser*.

Setelah pemain memasukan kode, maka halaman web di *PC* dan *smartphone* akan berubah. Pada *PC*, halaman akan menunjukan berbagai jenis permainan yang dapat dipilih. Pada *smartphone*, halaman akan berubah menjadi pengendali permainan, dimana pemain dapat menggerakan halaman yang ada di *PC* dengan menggunakan *smartphone*.

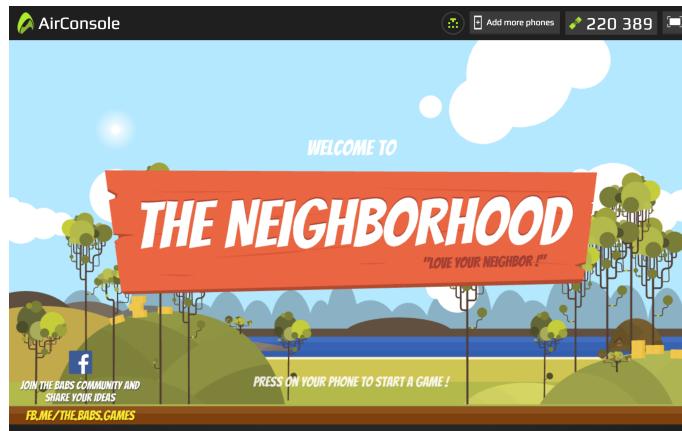


Gambar 3.6: Halaman pada *PC* yang menunjukan berbagai permainan yang dapat dipilih.



Gambar 3.7: Halaman pada *smartphone* yang berfungsi sebagai pengendali.

Dalam analisis ini, penulis memilih untuk memainkan permainan yang bernama *The Neighborhood*. Permainan ini sejenis permainan *Angry Birds*. Permainan ini bercerita tentang dua kelompok yang bertetangga, dimana kelompok tersebut bermusuhan dan berusaha untuk saling menghancurkan satu sama lain. Tujuan dari permainan ini yaitu lebih dulu menghancurkan anggota kelompok tetangga. Setelah memilih permainan tersebut, halaman pada *PC* dan *smartphone* akan berubah. Pada *smartphone*, pemain akan diminta untuk merubah mode tampilan *smartphone* menjadi *landscape*.

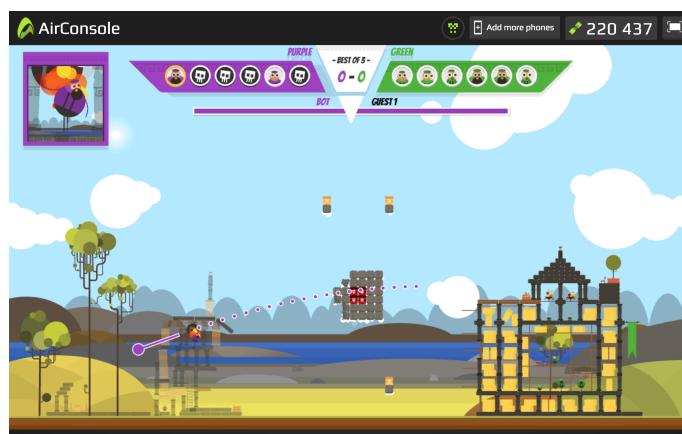


Gambar 3.8: Halaman awal permainan The Neighborhood pada *PC*.

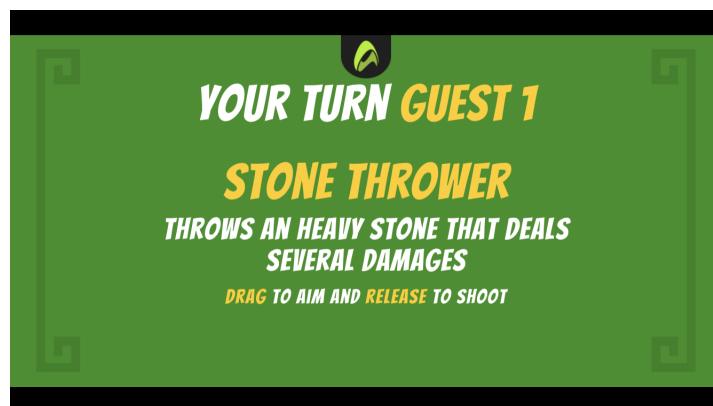


Gambar 3.9: Halaman awal permainan The Neighborhood pada *smartphone*.

Cara bermain dari permainan tersebut yaitu dengan menggunakan *smartphone*, dimana pemain harus menekan layar *smartphone*, kemudian menariknya sesuai dengan arah yang berlawanan dengan lawan, lalu melepas jari dari layar *smartphone* dengan tujuan untuk melempar suatu benda dari ketapel. Semakin jauh pemain menarik, maka lontaran benda tersebut akan semakin kencang mengenai lawan.

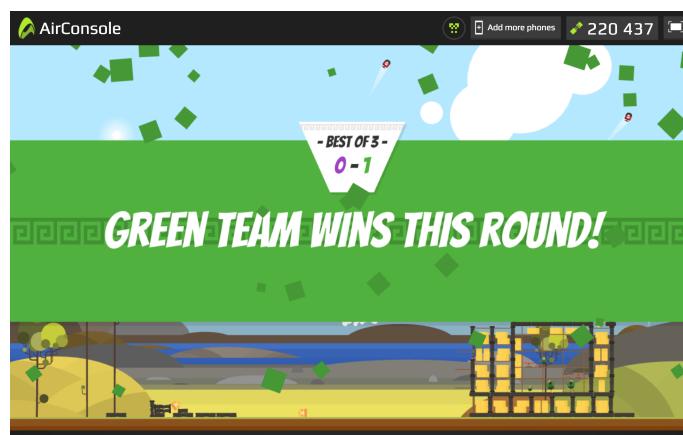


Gambar 3.10: Halaman pada *PC* dimana permainan sedang berlangsung.

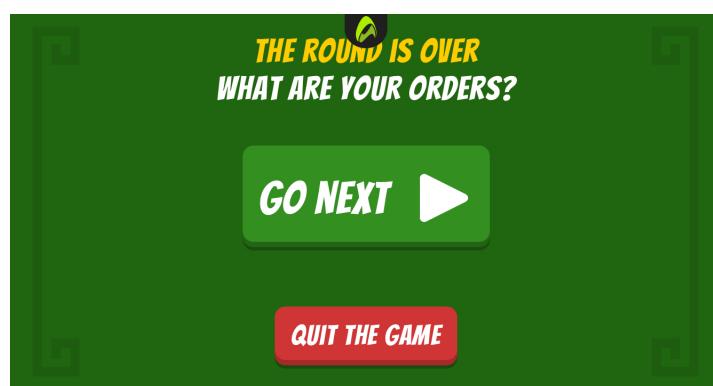


Gambar 3.11: Halaman pada *smartphone* dimana permainan sedang berlangsung.

Apabila memenangkan permainan tersebut, maka pemain dapat memilih untuk keluar dari permainan atau melanjutkan permainannya kembali.



Gambar 3.12: Halaman pada *PC* apabila permainan sudah dimenangkan.



Gambar 3.13: Halaman pada *smartphone* apabila permainan sudah dimenangkan.

Dari ketiga percobaan yang sudah dilakukan, ada beberapa hal yang dapat diperbaiki dari permainan berbasis web tersebut. Percobaan pertama menunjukkan hasil yang bagus, dimana koneksi antara *smartphone* dan *PC* tidak putus saat permainan berlangsung, dan juga tidak ada keterlambatan antara gerakan pada *smartphone* dan *PC*. Pada percobaan kedua, apabila *browser* pada *PC* ditutup pada saat permainan berlangsung, maka koneksi akan terputus. Tetapi, tampilan pada *smartphone* tidak menunjukkan bahwa adanya koneksi yang terputus, sehingga pemain tidak

mengetahui apakah permainan masih dapat berlangsung atau tidak. Tampilan hanya akan langsung kembali pada halaman awal permainan. Begitupun dengan percobaan ketiga, apabila *browser* pada *smartphone* ditutup pada saat permainan sedang berlangsung, maka koneksi akan terputus. Tampilan pada *PC* hanya menunjukkan tanda kecil bahwa telah terjadi pemutusan koneksi pada *smartphone*, yaitu tanda x pada bagian atas tampilan yang ditunjukkan seperti gambar berikut:



Gambar 3.14: Halaman pada *PC* yang menunjukkan pemutusan koneksi.

## 3.2 Analisis Alur Permainan Finger For Life

Web yang akan dikembangkan merupakan aplikasi permainan berbasis web yang bernama Finger For Life. Jenis permainan ini adalah kompetisi balap lari yang dilakukan oleh dua pemain. Tujuan dari permainan ini adalah mencapai garis akhir lintasan lari lebih dulu untuk menjadi pemenang. Untuk dapat memainkan permainan ini, para pemain harus menyiapkan beberapa hal seperti berikut:

- Satu Komputer *PC*.
- Dua *smartphone*.
- Jaringan Internet.

Para pemain harus menyediakan satu perangkat komputer *PC*, dimana *PC* tersebut akan berfungsi sebagai *console*, dan dua *smartphone* yang akan berfungsi sebagai *controller*. *PC* akan memiliki peran sebagai *host* yang menciptakan suatu *room*, dimana *room* tersebut merupakan ruang permainan yang dapat dimainkan. Ketiga perangkat tersebut akan membutuhkan jaringan internet untuk dapat mengakses alamat web yang harus dituju. Setelah ketiga perangkat tersebut tersedia, maka para pemain dapat membuka *browser* untuk mengakses alamat web permainan Finger For Life. Agar para pemain dapat mulai memainkan permainan ini, ada beberapa tahap yang harus dilakukan. Beberapa tahap yang dilakukan akan dibagi menjadi dua, yaitu yang terjadi pada bagian *PC*, dan pada *smartphone*. Tahap-tahap tersebut akan dijelaskan sebagai berikut:

### 1. Permintaan Bergabung

#### **PC**

Saat *client* mengakses alamat web Finger For Life pada *PC*, *client* akan ditujukan kehalaman permintaan bergabung. Pada tahap ini, halaman pada *PC* akan menunjukkan kode yang harus dikirimkan oleh para pemain. Kode tersebut merupakan *room* yang akan menyimpan tiga *client* pada saat permainan dimulai. Ketiga *client* tersebut adalah *host*, pemain pertama, dan pemain kedua. Para pemain harus mengirimkan kode tersebut kepada *host*, sehingga *smartphone* pemain dapat tersambung dengan *PC*. Kode yang tersedia berfungsi untuk proses

verifikasi, apakah *client* dapat bergabung kedalam *room* atau tidak.

#### **Smartphone**

Pada halaman *smartphone*, *client* harus mengisi form yang tersedia dengan kode *room* dari *PC*. Setelah kode dikirimkan, maka *client* akan segera mendapatkan pesan yang menandakan apakah sudah berhasil bergabung kedalam *room* atau tidak. Apabila pemain yang bergabung masih berjumlah satu, maka *room* masih menunggu untuk pemain lainnya bergabung. *Room* akan ditutup apabila pemain yang bergabung sudah berjumlah dua. Setelah tahap ini selesai, maka akan ditujukan ke halaman lain untuk tahap berikutnya.

### **2. Memilih Karakter**

#### **PC**

Para pemain yang telah bergabung akan segera ditujukan kehalaman memilih karakter. Pada halaman ini, akan muncul karakter yang telah dipilih oleh para pemain. Pada tahap ini, kedua pemain harus menetapkan karakter yang merepresentasikan para pemain dalam permainan. Apabila kedua pemain belum menetapkan karakter, maka tidak dapat menuju ke tahap selanjutnya.

**Smartphone** Halaman ini akan menunjukkan daftar karakter yang dapat dipilih oleh para pemain. Karakter yang dipilih akan ditunjukkan dilayar *PC*. Apabila pemain akan menetapkan karakter yang dipilih, maka pemain dapat menekan tombol *choose*. Setelah kedua pemain menetapkan karakter masing-masing, maka akan dituju kehalaman selanjutnya.

### **3. Memainkan Permainan**

#### **PC**

Permainan sudah dapat dimainkan pada tahap ini. Sebelum permainan dimulai, halaman *PC* akan menampilkan *countdown* selama tiga detik. Setelah waktu tiga detik selesai, maka akan ditampilkan lintasan lari yang harus dilalui para pemain, dan dua karakter yang sudah ditetapkan oleh kedua pemain. Karakter tersebut akan berlari melalui lintasan untuk mencapai garis akhir. Pemain yang lebih dulu mencapai garis akhir akan menjadi pemenang.

#### **Smartphone**

Pada saat *countdown* dilakukan halaman *PC*, halaman *smartphone* akan menampilkan instruksi untuk memainkan permainan ini. Instruksi tersebut menunjukan untuk mengubah mode tampilan *smartphone* dari *portrait* menjadi *landscape*. Setelah *countdown* selesai, maka halaman akan menampilkan dua telapak kaki yang berfungsi sebagai tombol. Untuk dapat menggerakan karakter, pemain harus menekan kedua tombol tersebut secara berulang. Apabila sudah ada pemain yang mencapai garis akhir, maka halaman akan dituju kehalaman selanjutnya.

### **4. Mengakhiri Permainan**

#### **PC**

Halaman ini akan menampilkan pemenang yang berhasil mencapai garis akhir lebih dulu. Halaman ini menampilkan podium yang menempatkan para pemenang sesuai posisi nomor urut masing-masing. Pada tahap ini, karakter milik pemenang akan ditampilkan di podium nomor satu, dan pemain selanjutnya dinomor dua. Untuk mengakhiri permainan, pemain harus menekan tombol *exit* yang terdapat dihalaman ini.

#### **Smartphone**

Pada halaman ini akan ditampilkan teks yang menunjukan posisi pemenang. Apabila pemain lebih dulu mencapai garis akhir, maka akan ditampilkan teks yang menunjukan posisi pertama. Setelah tombol *exit* dihalaman *PC* ditekan, maka permainan telah berakhir, dan halaman pada *PC* dan *smartphone* akan menuju kehalaman awal.

### 3.3 Analisis Pengembangan Web

Penjelasan pustaka-pustaka yang dibutuhkan untuk pengembangan Finger For Life sudah dijelaskan pada bab 2. Pada subbab ini akan dijelaskan bagaimana pustaka-pustaka tersebut digunakan dalam proses implementasi aplikasi permainan berbasis web ini.

#### 1. Node.js

- (a) **HTTP** HTTP API digunakan pada bagian *server*. Web yang akan dikembangkan akan menggunakan HTTP *server* untuk menyediakan akses yang dapat dilakukan oleh *client*. HTTP *server* akan diintegrasikan dengan Socket.io, dimana HTTP *server* akan menjadi *input* parameter yang dibutuhkan Socket.io.

Contoh penggunaan HTTP *server* akan dijelaskan sebagai berikut:

```
var express = require('express');
var http = require('http');
var socketIO = require('socket.io');

var app = express();
var httpServer = http.createServer(app);
var io = socketIO(server);
```

Bagian pertama potongan kode merupakan proses yang dilakukan untuk mendapatkan fungsi-fungsi yang tersedia dari masing-masing modul. Dengan melakukan proses tersebut, maka dapat dibuat variabel baru untuk melakukan proses inisialisasi yang dilakukan pada bagian kedua potongan kode.

Variabel *app* menginisialisasi Express.js yang akan melakukan penanganan fungsi *server*. Variabel tersebut akan disambungkan dengan HTTP *server*, dengan menggunakan variabel *server*. Setelah variabel *httpServer* diinisialisasi, variabel *io* dapat diintegrasikan dengan HTTP *server* agar dapat menggunakan Socket.io sebagai *server*. Dengan melakukan proses ini, maka *client* dapat terkoneksi dengan Socket.io.

Salah satu *method* yang dimiliki oleh HTTP adalah sebagai berikut:

- **http.createServer([options][,requestListener])**

Pada pengembangan Finger For Life, *method* ini digunakan untuk membangun *server* HTTP. Berikut merupakan contoh penggunaan dalam pengembangan Finger For Life:

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

var app = express();
var httpServer = http.createServer(app);
```

Baris pertama dan kedua pada potongan kode merupakan proses untuk mendapatkan fungsi dari masing-masing modul. Baris keempat merupakan proses inisialisasi Express.js yang dilakukan variabel *app*. Baris selanjutnya merupakan proses menciptakan HTTP *server* dengan menyambungkan variabel *app* kedalam parameter.

- (b) **Path**

Path digunakan untuk mengatur akses dari suatu direktori dan berkas didalam pengembangan Finger For Life. Salah satu *method* yang dimiliki oleh modul ini adalah sebagai berikut:

- **path.join(...path)**

Berikut merupakan contoh implementasi dari *method* ini:

```
path.join(__dirname + 'public');
```

*Method* ini menerima parameter **\_\_dirname**, yang merepresentasikan lokasi direktori dari berkas saat ini yang sedang dimanipulasi. Parameter tambahan yang diterima *method* ini adalah **public**, yang merepresentasikan nama direktori yang akan disambungkan dengan parameter sebelumnya.

### (c) Module

Dalam pengembangan Finger For Life, Module digunakan untuk memberikan akses pada direktori atau berkas lain untuk mendapatkan fungsi dari satu berkas tertentu. Contoh implementasi dari Module adalah sebagai berikut:

```
module.exports = app;
```

Potongan kode ini akan diletakan di bawah suatu berkas. Dengan melakukan proses ini, maka berkas dan direktori lain akan dapat menggunakan fungsi-fungsi dari modul tersebut .

## 2. Express.js

### (a) express()

- **express.static(root, [options])**

Pada pengembangan Finger For Life, *method* ini digunakan untuk menyediakan akses kepada suatu modul agar dapat mengakses modul-modul lainnya. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
express.static(path.join(__dirname, 'public'))
```

*Method* ini akan menerima parameter yang merepresentasikan direktori yang bersifat **static**. Dengan melakukan proses ini, maka modul saat ini akan dapat mengakses setiap modul yang ada pada direktori **public**.

- **express.Router([options])**

Pada pengembangan Finger For Life, *method* ini digunakan untuk mendapatkan fungsi yang dimiliki oleh modul Router. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
var router = express.Router();
```

Potongan kode tersebut menunjukkan proses inisialisasi dari variabel *router* untuk mendapatkan fungsi yang dimiliki oleh modul **Router**.

### (b) Application

- **app.set(name, value)**

Pada pengembangan Finger For Life, *method* ini digunakan untuk menetapkan suatu nilai tertentu kepada parameter **name**. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
app.set('view engine', 'ejs');
```

Potongan kode tersebut menunjukkan bahwa nilai *ejs* akan ditetapkan kepada *view engine*. Dengan begitu, *view engine* akan mengembalikan *ejs* apabila diperlukan untuk diakses.

- **app.use([path,] callback[, callback...])**

Pada pengembangan Finger For Life, *method* ini digunakan untuk menetapkan fungsi untuk menangani *path* yang telah ditentukan. Apabila pengguna mengakses *path* '/', maka *homeRouter* akan dieksekusi. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
app.use('/', homeRouter);
```

Potongan kode tersebut menunjukan bahwa apabila suatu *client* mengakses *path* yang tersedia, maka Express.js akan mengarahkan halaman web menuju *homeRouter*.

### 3. Socket.io

#### (a) Server API

Berikut merupakan beberapa kelas yang dimiliki oleh *Server API*:

##### i. Server

- **new Server(*httpServer*[, *options*])**

Pada pengembangan Finger For Life, konstruktor dari kelas ini dipakai untuk membuat *server* Socket.io berdasarkan *server* HTTP milik Node.js. Contoh implementasi dari konstruktor ini adalah sebagai berikut:

```
const http = require('http');
const socketIO = require('Socket.io');
const express = require('express');
```

```
var app = express();
var httpServer = http.createServer(app);
var io = socketIO(httpServer);
```

Bagian pertama potongan kode merupakan proses yang dilakukan untuk mendapatkan fungsi-fungsi yang tersedia dari masing-masing modul. Dengan melakukan proses tersebut, maka dapat dibuat variabel baru untuk melakukan proses inisialisasi yang dilakukan pada bagian kedua potongan kode.

Variabel *app* menginisialisasi Express.js yang akan melakukan penanganan fungsi *server*. Variabel tersebut akan disambungkan dengan HTTP *server*, dengan menggunakan variabel *server*. Setelah variabel *httpServer* diinisialisasi, variabel *io* dapat diintegrasikan dengan HTTP *server* agar dapat menggunakan Socket.io sebagai *server*. Dengan melakukan proses ini, maka *client* dapat terkoneksi dengan Socket.io.

##### ii. Namespace

- **namespace.emit(*eventName*[, ...*args*])**

Pada pengembangan Finger For Life, *method* ini digunakan untuk memancarkan suatu *event* apabila terjadi aksi yang dilakukan oleh *client* maupun *server*. Dengan menggunakan *method* ini, *event* akan dipancarkan ke seluruh *client* yang terhubung kepada Socket.io. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
io.emit('buttonClicked', 'Send this to every client');
```

*Method* tersebut akan memancarkan *event buttonClicked* kepada seluruh *client*, dan mengirimkan pesan berupa teks yang bertuliskan *Send this to every client*.

- **namespace.to(*room*)**

Pada pengembangan Finger For Life, *method* ini akan digunakan untuk memancarkan suatu *event* kepada *client* yang hanya berada didalam *room* tertentu. *Client* yang tidak berada didalam *room* tidak akan mendapatkan *event* yang dipancarkan. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
io.to('928799').emit('requestAccepted', message);
```

*Method* ini akan memancarkan *event requestAccepted* kepada *client* yang berada didalam *room* 928799. *Event* tersebut akan mengirimkan data yang direpresentasikan oleh parameter *message*.

iii. **Socket** Beberapa properti yang dimiliki oleh kelas ini adalah sebagai berikut:

- **socket.id**

Pada pengembangan Finger For Life, properti ini berfungsi untuk mendapatkan identifikasi unik Socket.io milik setiap *client* yang sudah melakukan koneksi dengan Socket.io. Contoh implementasi dari properti ini adalah sebagai berikut:

```
var socketID = socket.id
```

Variabel ini akan mendapatkan identifikasi unik Socket.io yang dimiliki oleh *client*.

- **socket.room**

Pada pengembangan Finger For Life, properti ini digunakan untuk mengidentifikasi *client* berdasarkan *room* yang dimasukinya. Contoh implementasi dari properti ini adalah sebagai berikut:

```
var myRoom = socket.room
```

Variabel ini akan mendapatkan nilai *room* yang dimiliki oleh *client*.

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

- **socket.join(room[, callback])**

Pada pengembangan Finger For Life, *method* ini berfungsi untuk menambahkan *client* kedalam *room* tertentu. Pada aplikasi web yang akan dibangun, permainan dapat dimainkan oleh beberapa *client*. Untuk dapat membedakan pasangan-pasangan pemain yang sedang bermain, digunakan *room* agar terbagi menjadi beberapa ruang dalam permainan. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
socket.join('room305');
```

*Method* ini akan menambahkan *client* kedalam *room305*, yang nantinya dapat diidentifikasi melalui properi *socket.room*.

## (b) Client API

### i. IO

- **io([url][, options])**

Pada pengembangan Finger For Life, *method* ini digunakan agar *client* dapat menggunakan fungsi-fungsi yang dimiliki oleh Socket.io. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
var socket = io();
```

Variabel ini akan menginisialisasi Socket.io *client*, sehingga fungsi-fungsi yang tersedia dapat digunakan.

### ii. Socket

- **socket.on(eventName, callback)**

Pada pengembangan Finger For Life, *method* ini digunakan untuk menyediakan fungsi yang akan menangani *event* yang dipancarkan. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
socket.on('sendEvent', function(){
    console.log('An event has been sent');
});
```

*Method* ini akan menyediakan fungsi untuk yang akan menangani saat *event sendEvent* dipancarkan. Fungsi tersebut akan menampilkan pesan *And event has been sent*.

- **socket.emit(eventName[, ..args][, ack])**

Pada pengembangan Finger For Life, *method* ini digunakan untuk memancarkan suatu *event* kepada satu *socket* saja. *Event* hanya akan dipancarkan kepada *server* Socket.io. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
socket . emit ( 'sendMessage' , 'It works.' );
```

*Method* ini akan memancarkan *event sendMessage* dengan mengirimkan pesan *It works.*

Beberapa *event* yang dimiliki oleh kelas ini adalah sebagai berikut:

- **connect**

Pada pengembangan Finger For Life, *event* ini berfungsi untuk menangani koneksi yang dilakukan oleh *client* apabila sudah berhasil terhubung ke Socket.io. Contoh implementasi dari *event* ini adalah sebagai berikut:

```
socket . on ( 'connect' , function () {
    console . log ( 'Connected to the server!' );
});
```

*Event* ini akan menampilkan suatu pesan berupa teks yang menunjukkan sudah berhasil terkoneksi ke Socket.io, apabila *client* telah melakukan koneksi ke Socket.io.

- **disconnect**

Pada pengembangan Finger For Life, *event* ini digunakan untuk menangani koneksi Socket.io yang terputus. Contoh implementasi dari *event* ini adalah sebagai berikut:

```
socket . on ( 'disconnect' , function () {
    console . log ( 'Disconnected from the server.' );
});
```

*Event* ini akan menampilkan suatu pesan berupa teks yang menunjukkan koneksi yang terputus ke Socket.io.

## 4. Canvas API

### (a) Animation

- **setInterval(func, delay[, param1, param2, ...])**

Pada pengembangan Finger For Life, *method* ini digunakan untuk menampilkan animasi dengan cara mengeksekusi fungsi yang ada selama waktu *delay* yang telah ditentukan. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
setInterval ( showCountDown , 1000 );
```

*Method* ini akan mengeksekusi *method* *showCountDown* selama waktu delay 1000 *millisecond*.

- **clearInterval(intervalID)**

Pada pengembangan Finger For Life, *method* ini digunakan untuk memberhentikan pengulangan eksekusi *method* yang sedang dilakukan oleh *method setInterval()*. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
var timer = setInterval ( showCountDown , 1000 );
```

```
clearInterval ( timer );
```

*Method* ini akan mengakhiri pengulangan eksekusi variabel *timer*, yang merupakan *method* *setInterval()*.

- **requestAnimationFrame(callback)**

Pada pengembangan Finger For Life, *method* ini digunakan untuk melakukan animasi dengan interaksi dari pemain. Apabila pemain menekan tombol tertentu, maka *method* ini akan dieksekusi untuk melakukan proses animasi pada permainan. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
function readyPlayerOne(){
    ctx.globalCompositeOperation = 'destination-over';
    ctx.clearRect(0, 0, 900, 600);
    ctx.save();

    progressPlayer1 += 1;

    ctx.restore();
    ctx.drawImage(track, 0, 0);

    if (progressPlayer1 < 40) {
        aniFrame = requestAnimationFrame(readyPlayerOne);
    }
    else {
        progressPlayer1 = 0;
    }
}
```

*Method readyPlayerOne* digunakan untuk proses membuat animasi didalam permainan. Pada *method* ini akan dilakukan pengecekan pada variabel *progresPlayer1*, apabila masih bernilai kurang dari 40, maka *method requestAnimationFrame()* masih akan dieksekusi.

(b) **canvasRenderingContext2D**

Beberapa properti yang dimiliki oleh kelas ini adalah sebagai berikut:

- **CanvasRenderingContext2D.globalCompositeOperation**

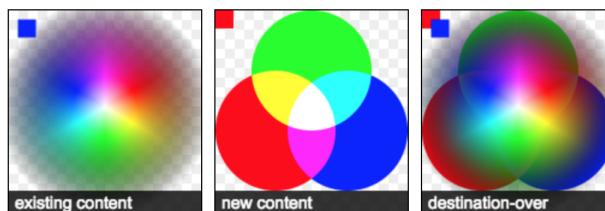
Pada pengembangan Finger For Life, properti ini digunakan untuk membuat animasi, dengan cara menimpa gambar sebelumnya dengan gambar baru. Salah satu properti yang digunakan dalam proses implementasi adalah sebagai berikut:

- **destination-over**

Contoh implementasi dari *properti* ini adalah sebagai berikut:

```
ctx.globalCompositeOperation = 'destination-over';
```

Properti ini akan menetapkan nilai *globalCompositeOperation* menjadi *destination-over*. Nilai tersebut akan memberikan efek seperti berikut:



Gambar 3.15: Efek *destination-over*

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

- **CanvasRenderingContext2D.clearRect(x, y, width, height)**

Pada pengembangan Finger For Life, *method* ini digunakan untuk menghilangkan gambar tertentu dari layar dengan tujuan untuk menampilkan gambar baru kelayar. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
ctx.clearRect(0, 0, 900, 600);
```

*Method* ini akan menghilangkan seluruh isi dari *rectangle* pada *canvas*, yang dimulai dari posisi (0,0), dengan lebar 600, dan panjang 900.

- **CanvasRenderingContext2D.drawImage(image, dx, dy)**

Contoh implementasi dari *method* ini adalah sebagai berikut:

```
ctx.drawImage(image, 100, 100);
```

*Method* ini akan menggambar variabel *image* pada posisi (100,100).

- **CanvasRenderingContext2D.save()**

Pada pengembangan Finger For Life, *method* ini digunakan untuk menyimpan *state* dari *canvas* saat ini, sebelum *state* tersebut dihilangkan untuk menaruh gambar yang baru. Tujuan dari proses ini adalah untuk membuat animasi pada permainan. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
ctx.save();
```

- **CanvasRenderingContext2D.restore()**

Pada pengembangan Finger For Life, *method* ini digunakan untuk mengembalikan *state* dari *canvas* saat ini menjadi *state* sebelumnya. *Method* ini digunakan didalam proses animasi permainan. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
ctx.restore();
```

## 5. jQuery

### (a) **submit(handler)**

Pada pengembangan Finger For Life, *method* ini digunakan untuk mengirimkan data kode *room* yang sudah diisi oleh pengguna kedalam kolom. *Method* ini akan digunakan pada tahap permintaan bergabung bagi pengguna. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
$( 'form' ).submit( function(e){
    e.preventDefault();
    socket.emit('requestToJoin', {
        id: socket.id,
        room: $('#code').val()
    });
});
```

*Method* ini akan mengambil elemen *form* dari berkas HTML, untuk mengambil data yang telah diisi oleh pengguna yang kemudian dikirimkan ke *server*. Pada saat pengguna *smartphone* menekan tombol *send* pada layar, maka *method* ini akan dieksekusi.

### (b) **val()**

Pada pengembangan Finger For Life, *method* ini digunakan untuk mengambil nilai dari data yang telah diisi oleh pengguna kedalam kolom. *Method* ini akan digunakan pada tahap permintaan bergabung bagi pengguna. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
var code = $('#code').val()
```

*Method* ini akan mengakses elemen HTML dengan *id code*, kemudian mengambil data yang ada didalamnya untuk disimpan ke variabel *code*.

(c) **html()**

Pada pengembangan Finger For Life, *method* ini digunakan untuk mengambil seluruh isi dari elemen HTML. *Method* ini digunakan pada proses menampilkan seluruh halaman yang tersedia pada aplikasi web yang akan dibangun. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
var charMobileHtml = $("#charMobile").html();
```

*Method* ini akan mengakses elemen HTML yang memiliki *id charMobile*, yang kemudian akan mengambil seluruh isi dari elemen tersebut untuk disimpan ke variabel *charMobile-  
Html*.

(d) **preventDefault()**

Pada pengembangan Finger For Life, *method* ini digunakan untuk mencegah aksi *default* yang akan dilakukan *method submit()*, pada saat mengirim *form* ke *server*. *Method* ini dieksekusi pada saat pengguna mengirimkan data kode *room* untuk melakukan proses permintaan bergabung. Contoh implementasi dari *method* ini adalah sebagai berikut:

```
$( 'form' ).submit( function ( e ){
    e.preventDefault ();
});
```

*Method* ini akan mengakses elemen *form* dari HTML, dan mencegah aksi *default* dari *method submit()* dilakukan.

## 6. The Content Template element

Pada pengembangan Finger For Life, <template> digunakan untuk menyimpan seluruh halaman-halaman yang diperlukan untuk pengembangan aplikasi permainan berbasis web ini. Pada proses implementasi, terdapat banyak <template> yang menyimpan halaman web yang berbeda-beda. Contoh implementasi dari elemen ini adalah sebagai berikut:

```
<template id="homePage">
<div class="titleContainer">
<div class="leftFing">

</div>

<div class="wordContainer">
<div class="word">
<p>FINGER <br> FOR <br> LIFE</p>
</div>

<button id="startButton" onclick="startClicked()">START</button>
<button id="joinButton" onclick="joinClicked()">JOIN</button>

</div>

<div class="rightFing">

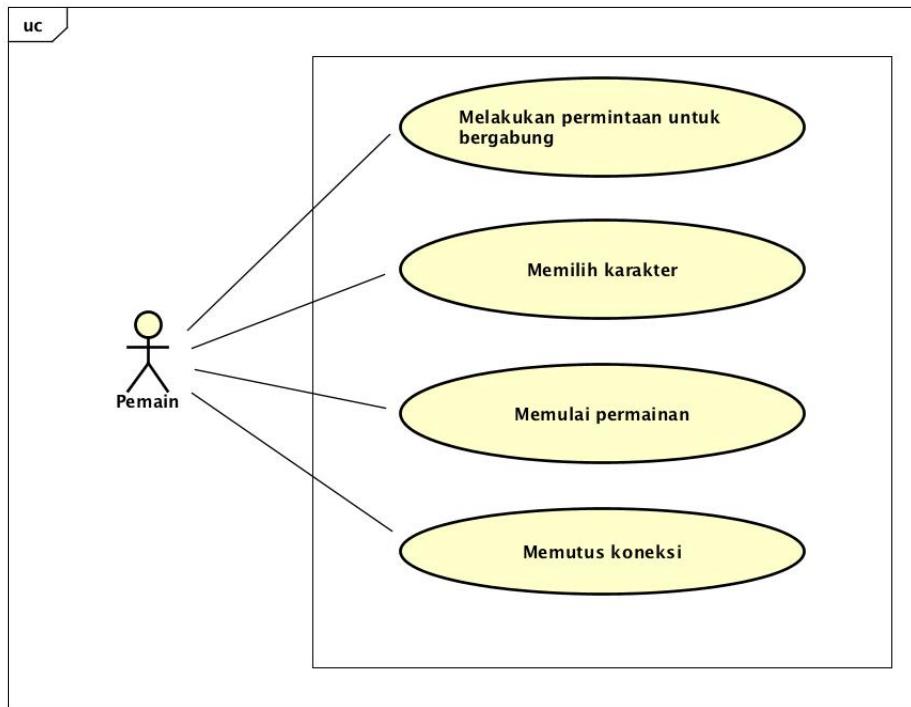
</div>
</div>
</template>
```

Elemen ini memiliki *id* dengan nama *homePage*, dimana elemen ini dapat diakses dengan menggunakan jQuery, apabila perlu ditampilkan kelayar pengguna.

## 3.4 Analisis *Use Case*

### 3.4.1 Diagram *Use Case*

Diagram *use case* pada permainan berbasis web yang akan dibangun hanya mengandung satu aktor, yaitu pemain. Diagram *use case* dapat dilihat pada Gambar 3.16.



Gambar 3.16: Diagram *use case* pemain

Berdasarkan hasil analisis pada subbab 3.2, akan dibentuk empat *use case* antara lain:

- **Melakukan permintaan untuk bergabung**, pemain dapat bergabung dalam permainan dengan mengirimkan kode yang sudah disediakan *browser PC* dengan menggunakan *browser smartphone*.
- **Memilih karakter**, pemain dapat memilih karakter yang tersedia di *browser smartphone*.
- **Memainkan permainan**, pemain dapat memainkan permainan dengan menekan tombol-tombol yang ada di *smartphone*.
- **Mengakhiri permainan**, pemain dapat mengakhiri permainan dengan memutus koneksi *Socket.io*.

### 3.4.2 Skenario *Use Case*

#### 1. Melakukan Permintaan Untuk Bergabung

- Nama: Melakukan permintaan untuk bergabung
- Aktor: Pemain

- Deskripsi: Melakukan permintaan untuk bergabung dengan *room* yang sudah tersedia
- Kondisi awal: Pemain telah membuka halaman *join* pada *browser smartphone* dan mengisi *form* dengan kode yang telah disediakan
- Kondisi akhir: Pemain bergabung kedalam *room*
- Skenario utama:

No	Aksi Aktor	Reaksi Sistem
1	Pemain mengisi <i>form</i> dengan kode <i>room</i> dan menekan tombol <i>send</i>	Sistem mendapatkan kode <i>room</i> dan memproses kode tersebut, kemudian memberikan <i>feedback</i> kepada pemain.

- Eksepsi: Pemain bergabung kedalam *room*.

## 2. Memilih Karakter

- Nama: Memilih karakter
- Aktor: Pemain
- Deskripsi: Memilih karakter yang ditampilkan pada *browser smartphone*
- Kondisi awal: Pemain telah bergabung kedalam *room*
- Kondisi akhir: Pemain menetapkan karakter yang akan dimainkan
- Skenario utama:

No	Aksi Aktor	Reaksi Sistem
1	Pemain memilih karakter yang tersedia dan menetapkan karakter tersebut.	Sistem menerima karakter yang dipilih dan menetapkan karakter yang dipilih oleh pemain

## 3. Memainkan Permainan

- Nama: Memainkan permainan
- Aktor: Pemain
- Deskripsi: Memainkan permainan dengan menekan tombol-tombol pada *browser smartphone* untuk menggerakkan karakter
- Kondisi awal: Pemain telah menetapkan karakter permainan
- Kondisi akhir: Pemain memainkan permainan dengan menggerakkan karakter
- Skenario utama:

No	Aksi Aktor	Reaksi Sistem
1	Pemain menekan tombol telapak kaki berulang-ulang	Sistem menangkap <i>event</i> menekan tombol telapak kaki dan menggerakan karakter milik pemain

## 4. Memutus Koneksi

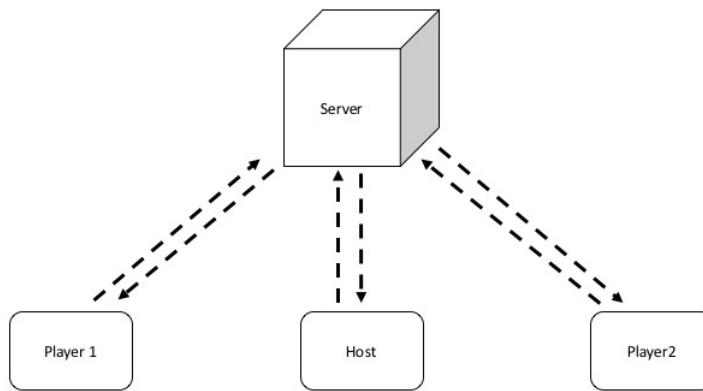
- Nama: Memutus koneksi
- Aktor: Pemain
- Deskripsi: Memutus koneksi dengan keluar dari *browser* atau menekan tombol *exit* saat telah selesai bermain

- Kondisi awal: Pemain telah selesai bermain
- Kondisi akhir: Pemain keluar dari permainan dan koneksi terputus
- Skenario utama:

No	Aksi Aktor	Reaksi Sistem
1	Pemain menekan tombol <i>exit</i> pada halaman terakhir atau menutup <i>browser</i>	Sistem akan menangkap event tombol <i>exit</i> ditekan dan memutus koneksi pemain

### 3.5 Analisis Aristektur Finger For Life

Arsitektur Finger For Life dapat dilihat pada Gambar 3.17 . Jumlah *Client* dalam permainan ini adalah tiga. *Client* tersebut antara lain *host*, *player 1*, dan *player 2*. Apabila *player 1* akan mengirimkan data untuk diproses oleh *host*, maka data tersebut akan dikirimkan terlebih dahulu kepada *server*, kemudian dilanjutkan kepada *host*.



Gambar 3.17: Arsitektur Finger For Life

Komunikasi antara *client* dengan *client* tidak dapat dilakukan secara langsung. Komunikasi tersebut harus melewati *server* terlebih dahulu.

Finger For Life dibangun diatas Node.js. Seluruh berkas serta folder yang dibutuhkan untuk membangun aplikasi permainan ini diatur dengan menggunakan Express.js. Proses pengiriman dan penerimaan data secara *realtime* dilakukan menggunakan Socket.io. Dalam permainan ini, animasi akan dilakukan dengan menggunakan Canvas API.

### 3.6 Analisis Socket.io

Pada pengembangan Finger For Life digunakan banyak fitur yang telah disediakan oleh Socket.io. Teknologi ini digunakan untuk proses sinkronisasi antara *smartphone* dengan *PC* dan komunikasi secara *real-time* antara *client* dan *server*. Seperti yang sudah dijelaskan pada subbab 3.2, ada beberapa tahap yang harus dilakukan sebelum dapat memainkan aplikasi permainan berbasis web ini. Tahap tersebut banyak menggunakan pustaka Socket.io dalam proses implementasinya.

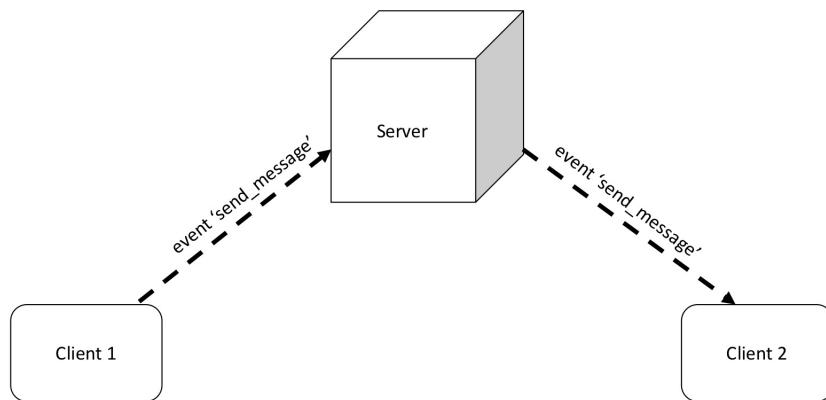
Subbab ini akan menjelaskan bagaimana teknologi Socket.io digunakan dalam implementasi Finger For Life pada tahap-tahap yang harus dilakukan.

#### 1. Komunikasi antara *client* dan *server*

Pada aplikasi permainan berbasis web ini, akan banyak dibutuhkan proses komunikasi antara

sesama *client* maupun dengan *server*. Komunikasi yang dilakukan harus secara *real-time*, agar interaksi didalam permainan dapat berjalan pada saat permainan dimainkan.

Interaksi tersebut akan dilakukan berdasarkan *event*. Suatu *event* akan dipancarkan kedalam suatu *stream*, dimana *stream* akan berisi berbagai *event* yang telah dipancarkan, yang menunggu untuk ditangkap oleh fungsi tertentu. Pada saat *client* akan berkomunikasi dengan *client* lain, maka *event* yang dipancarkan akan ditangkap oleh *server* sebelum kemudian dilanjutkan kepada *client* lain. Arsitektur interaksi antara *client* dan *server* dapat dilihat pada Gambar 3.18.



Gambar 3.18: Arsitektur interaksi *client* dan *server*

Agar proses tersebut dapat dilakukan, maka fitur yang dimiliki oleh Socket.io akan digunakan. Fitur-fitur tersebut adalah sebagai berikut:

- **socket.emit(eventName[, ...args][, ack])**

*Method* ini akan memancarkan *event* dengan nama *eventName*, dimana *event* tersebut dapat ditangkap oleh *client* maupun *server* dengan menggunakan fungsi tertentu. Apabila *event* yang telah dipancarkan tidak ditangkap oleh siapapun, maka *event* tersebut akan disimpan oleh *stream* hingga suatu sesi selesai yang kemudian akan dihancurkan.

- **socket.on(eventName, callback)**

*Method* ini akan menangkap *event* yang namanya sesuai dengan *eventName*. Setelah *event* tersebut ditangkap, maka fungsi *callback* akan dieksekusi.

Contoh implementasi dari proses interaksi antara *client* dan *server* adalah sebagai berikut:

```

\\ Client 1
socket . emit ( 'requestToJoin ' , {
    id: socket . id ,
    room: $( '#code ' ) . val ()
});

socket . on ( 'joinSucceed ' , function ( msg ){
    var messages = document . getElementById ( "joined " );
    messages . innerHTML = msg ;
})
  
```

```

});;

socket.on('joinRejected', function(msg){
    var messages = document.getElementById("joined");
    messages.innerHTML = msg;
});

```

Potongan kode ini menunjukan suatu *client* yang memancarkan *event requestToJoin* dengan data-data yang dikirimkan. *Event* tersebut akan ditangkap oleh *server* untuk dilakukan pengecekan terhadap data yang dikirimkan. Selain itu, *client* memiliki *event handler* yang akan menangkap *event joinRejected* dan *joinSucceed*.

```

\\Server
socket.on('requestToJoin', function(msg){
var check = users.isRoomExist(msg.room);

if (check === 1) {
io.to(msg.room).emit('requestAccepted', 'Client has joined the room.');
} else{
socket.emit('joinRejected', 'The room is not exist');
}
});

```

Pada potongan kode ini, *server* akan menangkap *event requestToJoin* yang dipancarkan oleh *client*. Apabila *client* telah memancarkan *event* tersebut, maka *server* akan menangkap *event* dan mengeksekusi fungsi *callback*. *Server* kemudian akan melakukan pengecekan terhadap data *msg* yang diterima. Apabila pengecekan menghasilkan angka satu, maka *server* akan memancarkan *event requestAccepted* kepada *client* lain yang berada didalam *room*. Apabila pengecekan tidak menghasilkan angka satu, maka *server* akan memancarkan *event joinRejected* kepada satu *socket* yang sedang melakukan koneksi kepada *server*.

```

\\Client 2
socket.on('requestAccepted', function(msg){
    showMessage(msg);
});

```

Potongan kode ini menunjukan suatu *client* yang akan menangkap *event requestAccepted* yang dikirimkan oleh *server*. *Client* kemudian akan mengeksekusi fungsi *callback* untuk mengolah data yang dikirimkan.

## 2. Sinkronisasi *PC* dan *Smartphone*

Pada permainan Finger For Life terdapat tahap permintaan bergabung yang dilakukan oleh *PC* dan *smartphone*. Tahap tersebut merupakan proses sinkronisasi yang harus dilakukan agar *smartphone* dan *PC* dapat tersambung dan dapat berkomunikasi satu sama lain. Kedua gawai tersebut dapat tersambung dengan menggunakan Socket.io. Ada beberapa langkah yang harus dilakukan, langkah-langkah tersebut antara lain:

- **Koneksi Socket.io**

Sebelum *PC* dan *smartphone* tersambung, kedua gawai tersebut harus tersambung dengan Socket.io. Proses melakukan koneksi terhadap Socket.io dilakukan pada saat halaman web mengakses halaman *sync*. Pada saat kedua gawai mengakses halaman tersebut, proses yang terjadi adalah sebagai berikut:

- Didalam berkas *syncScript.js* dan *mobileScript.js*, terdapat variabel yang memulai *client* untuk melakukan koneksi kepada Socket.io. Variabel tersebut adalah sebagai berikut:

```
var socket = io();
```

Variabel *socket* akan mendapatkan fungsi-fungsi milik Socket.io yang telah tersedia, yang dapat dipakai pada bagian *client*. Apabila *PC* dan *smartphone* telah mengakses halaman *sync*, maka kedua gawai tersebut telah tersambung dengan Socket.io. Walaupun sudah tersambung, kedua gawai tersebut belum dapat berkomunikasi satu sama lain.

- **Sinkronisasi *PC* dan *Smartphone***

*PC* akan menyediakan kode yang harus dikirimkan oleh *smartphone*. Kode tersebut merupakan kode *room* yang dibutuhkan untuk proses sinkronisasi antara *PC* dan *smartphone*. Kode *room* akan dibangkitkan secara acak pada halaman *sync*. Kode tersebut dibangkitkan dengan cara seperti berikut:

```
function getRandInt(){
    text = Math.floor(Math.random() * (999999 - 111111)) + 111111;
    document.getElementById("roomId").innerHTML = text;
}
```

Kode yang akan dibangkitkan secara acak berjumlah enam digit angka, dimana angka tersebut berada didalam rentang 000000 hingga 999999. Pada saat kode *room* telah dibangkitkan, *PC* secara otomatis akan langsung bergabung kedalam *room* tersebut. Langkah tersebut dapat dilakukan seperti berikut:

```
function getRandInt(){
    text = Math.floor(Math.random() * (999999 - 111111)) + 111111;
    document.getElementById("roomId").innerHTML = text;

    var roomString = text.toString();

    socket.emit('hostJoinRoom', {
        id: socket.id,
        room: roomString
    });
}
```

*PC* akan memancarkan *event hostJoinRoom* dimana yang akan ditangkap oleh *server*. *Event* tersebut mengirimkan data berupa identifikasi unik milik *client*, dan kode *room* yang telah dibangkitkan. *Server* akan menangkap *event* tersebut dan menambahkan *host* kedalam *room* yang telah dibangun. Proses tersebut dilakukan sebagai berikut:

```
socket.on('hostJoinRoom', (msg) => {
    socket.join(msg.room);
    users.removeUser(msg.id);
    users.addUser(msg.id, msg.room);
});
```

*Event hostJoinRoom* akan ditangkap oleh *server* kemudian akan mengeksekusi fungsi yang ada didalamnya. *Server* akan menambahkan *host* kedalam *room* dengan menggunakan *socket.join(msg.room)*. Kemudian *host* akan ditambahkan kedalam *array* yang berada didalam kelas *Users*. *Array* tersebut berisi daftar *client* yang telah memiliki *room*. Proses tersebut dilakukan dengan menggunakan *users.addUser(msg.id, msg.room)*. Dengan melakukan proses ini, maka *room* telah dibangkitkan, dan *host* telah tergabung didalamnya.

Setelah kode *room* dibangkitkan, maka akan ditampilkan kelayar *PC*. Pengguna akan memasukan kode tersebut ke kolom yang berada dihalaman *smartphone* untuk dikirimkan

ke *server*. Proses ini merupakan proses sinkronisasi yang dibutuhkan antara *PC* dan *smartphone*. Proses pengiriman kode *room* akan dilakukan dengan cara berikut:

```
function requestToJoin(){
    $('form').submit(function(e){
        e.preventDefault();
        socket.emit('requestToJoin', {
            id: socket.id,
            room: $('#code').val()
        });
    });
}
```

*Smartphone* akan mengirimkan kode *room* dengan memancarkan *event requestToJoin*. Kode *room* yang telah diisi didalam kolom akan diambil dengan menggunakan jQuery. Proses tersebut dilakukan dengan menggunakan *jQuery('#code').val()*. *Event requestToJoin* akan mengirim data identifikasi unik Socket.io milik *client* dan kode *room*. *Event* tersebut akan ditangkap oleh *server* yang kemudian akan dilakukan proses otentikasi. Proses tersebut akan dilakukan seperti berikut:

```
socket.on('requestToJoin', function(msg){
    var check = users.isRoomExist(msg.room);

    //debug purpose
    console.log('this is check ${check}');

    if (check === 1) {
        socket.join(msg.room);
        users.removeUser(msg.id);
        users.addUser(msg.id, msg.room);

        io.to(user.room).emit('requestAccepted', msg);
        socket.emit('joinSucceed', 'Welcome to the game :)');
    } else{
        socket.emit('joinRejected', 'The room is not exist');
    }
})
```

Listing 3.1: Proses otentikasi

*Event* ini akan memeriksa apakah kode yang dikirimkan oleh *smartphone* sesuai dengan data *room* yang ada didalam *array*. Pengecekan tersebut dilakukan dengan menggunakan *var check = users.isRoomExist(msg.room)*. Apabila *room* sesuai, maka *smartphone* dapat bergabung kedalam *room*, apabila tidak sesuai, maka *smartphone* tidak dapat bergabung. Dengan bergabungnya *smartphone* kedalam *room*, maka proses sinkronisasi antara *smartphone* dan *PC* telah terpenuhi. Dengan begitu, kedua gawai tersebut dapat melakukan komunikasi satu sama lain.



## BAB 4

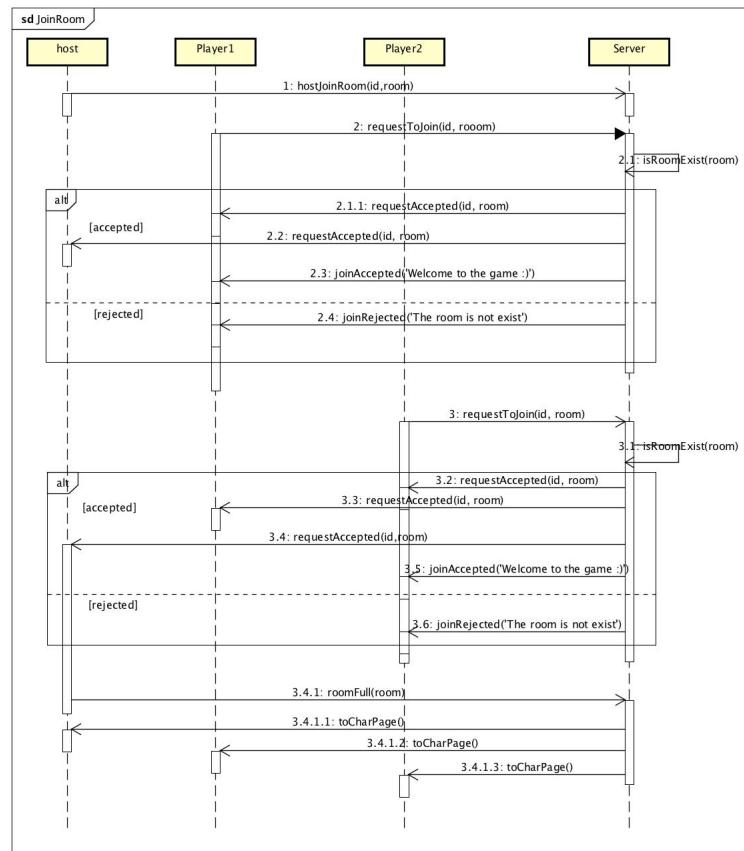
# PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan aplikasi yang akan dibangun meliputi diagram kelas rinci beserta deskripsi dan fungsinya, dan perancangan antarmuka.

#### 4.1 Perancangan Sequence Diagram

Pada *sequence diagram* akan dibahas mengenai jalannya koneksi *socket.io* dari awal koneksi mulai tersambung hingga koneksi terputus. *Sequence diagram* meliputi *sequence Join Room*, *sequence Choose Character*, *sequence Game Begin*, *sequence Winning Game*.

#### 4.1.1 *Sequence* Permintaan Bergabung



Gambar 4.1: Proses melakukan koneksi ke *socket.io* dan bergabung kedalam *room*.

Pada awal permainan, *client* pertama yang melakukan koneksi pada *server* adalah *PC computer*, yang berperan sebagai *host* dalam permainan seperti yang dijelaskan pada Gambar 4.1. *Host* akan

menyediakan suatu kode yang berguna sebagai *room* untuk kedua pemain yang akan bergabung dengan melakukan koneksi ke *server*. *Room* yang disediakan hanya akan menerima tiga *client* saja, yaitu *host*, *player1*, dan *player2*.

*Host* akan mengirimkan *event* yang menandakan akan bergabung kedalam room, *event* tersebut adalah **hostJoinRoom(id, room)**. *Event* ini memiliki data yang akan dikirimkan kepada *server*, data-data tersebut dijelaskan sebagai berikut:

- **id** identifikasi unik yang dimiliki masing-masing *client* yang terkoneksi dengan *socket.io*.
- **room** suatu *string* yang menandakan ruangan dimana *client* hanya akan melakukan komunikasi didalam ruangan tersebut.

Data-data tersebut akan dimasukan kedalam suatu *array*, dimana *array* tersebut akan menyimpan seluruh *client* yang terkoneksi dengan *socket.io*.

Setelah *host* terkoneksi dengan *socket.io*, maka *room* milik *host* sudah tersedia dan para pemain dapat bergabung kedalam *room* tersebut. Untuk dapat mulai bermain, para pemain harus memasukan kode *room* yang disediakan di halaman *host*. Pemain akan mengirimkan *event requestToJoin(id, room)* pada *server*. Data yang dikirimkan oleh pemain sama dengan data yang dikirimkan oleh *host*.

Setelah *event* tersebut diterima oleh *server*, maka akan dilakukan pengecekan apakah pemain dapat bergabung atau tidak kedalam *room*. Pengecekan tersebut dilakukan dengan menggunakan *method isRoomExist(room)*. Beberapa pengecekan yang dilakukan *method* ini akan dijelaskan sebagai berikut:

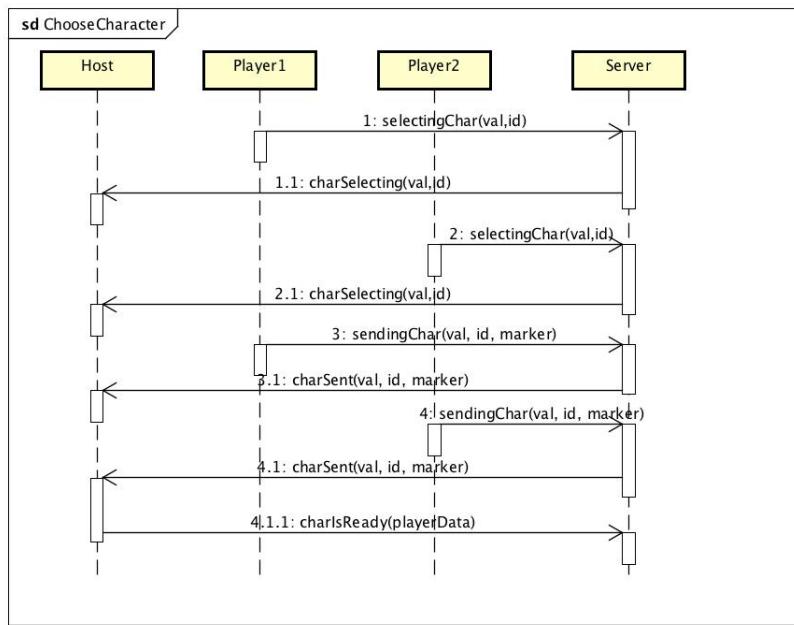
1. Memeriksa apakah data *room* didalam *array* ada yang sesuai dengan parameter *room*.
2. Memeriksa apakah jumlah yang ada didalam *room* yang dimaksud sudah lebih dari tiga atau belum.

Apabila kedua hal diatas terpenuhi, maka pemain akan terkoneksi ke *socket.io* dan bergabung ke dalam *room*. *Server* akan mengirimkan *event requestAccepted(id, room)* kepada seluruh *client* yang berada di *room* tersebut. *Event* tersebut berfungsi untuk mencatat *client* telah berhasil bergabung kedalam *room*. *Server* pun akan mengirimkan *event joinAccepted('Welcome to the game :)')* kepada pemain yang berhasil bergabung kedalam *room*. *Event* tersebut berfungsi untuk memberikan *feedback* kepada pemain bahwa pemain telah berhasil bergabung kedalam permainan.

Apabila kedua hal diatas tidak terpenuhi, maka pemain tidak dapat terkoneksi ke *socket.io*. *Server* akan mengirimkan *event joinRejected('The room is not exist')* kepada pemain. *Event* tersebut berfungsi sebagai *feedback* kepada pemain yang tidak dapat bergabung. Pemain pertama yang berhasil bergabung akan berperan sebagai *Player1*. Setelah satu pemain bergabung, maka *server* akan menunggu untuk pemain kedua agar permainan dapat dimulai. Pemain kedua akan melakukan hal yang sama dengan pemain pertama untuk dapat bergabung kedalam room.

Setelah *room* berisi tiga *client*, maka *host* akan mengirimkan *event roomFull(room)* kepada *server*. *Event* tersebut menandakan bahwa *room* sudah tidak dapat menerima *client* yang akan bergabung. Setelah *event* tersebut diterima oleh *server*, maka *event toCharPage()* akan dipancarkan oleh *server* kepada seluruh koneksi yang ada didalam *room* tersebut. *Event* ini berfungsi untuk mengganti halaman saat ini ke halaman selanjutnya.

### 4.1.2 Sequence Memilih Karakter



Gambar 4.2: Proses memilih karakter.

Halaman ini merupakan halaman yang dituju oleh para pemain yang telah berhasil melakukan koneksi dan bergabung kedalam *room*. Para pemain akan memilih karakter yang ada pada halaman *smartphone browser*, dimana karakter yang dipilih akan muncul pada halaman *PC browser*. Seperti yang dijelaskan pada Gambar 4.2, pemain yang memilih karakter tertentu akan mengirimkan event **selectingChar(val, id)** kepada *server*. Data-data yang dikirimkan akan dijelaskan sebagai berikut:

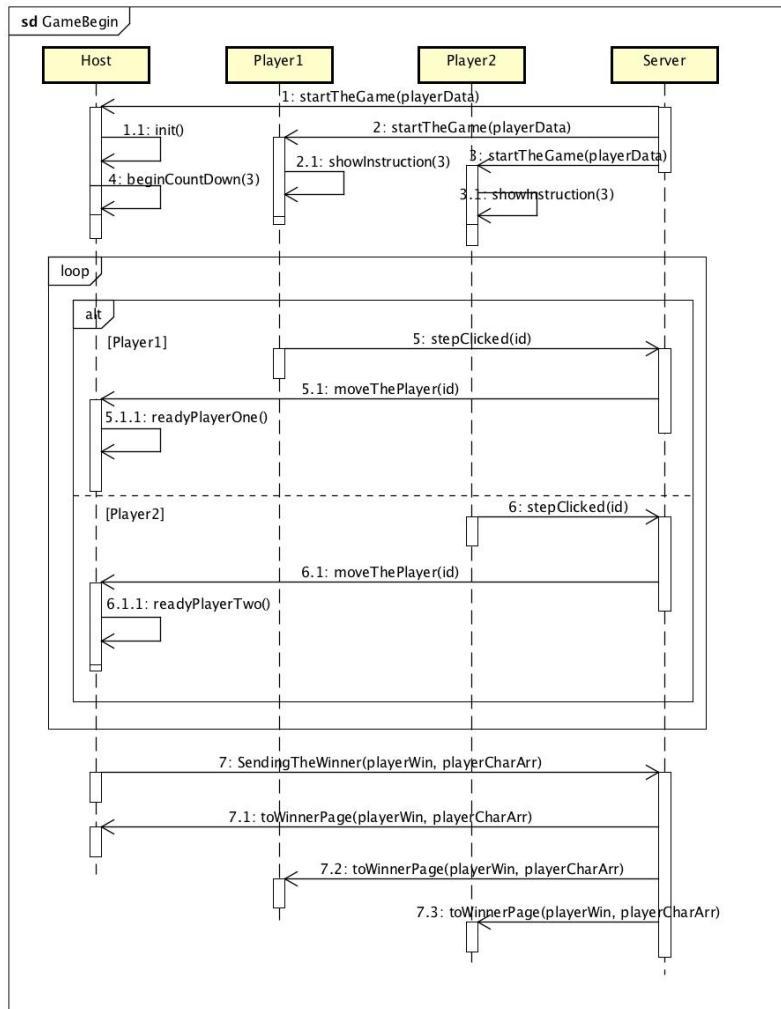
- **val** identifikasi unik yang dimiliki oleh masing-masing karakter.
- **id** identifikasi unik yang dimiliki masing-masing *client* yang terkoneksi dengan *socket.io*.

Setelah event tersebut diterima oleh *server*, maka server akan mengirimkan event **charSelecting(val, id)** kepada *host*. Event tersebut berfungsi untuk menampilkan karakter yang dipilih oleh pemain dihalaman *PC*.

Apabila pemain akan menetapkan karakter yang telah ditampilkan dihalaman *PC*, maka *pemain* akan mengirimkan event **sendingChar(val,id,marker)** kepada *server*. Data-data tersebut sama seperti yang dikirimkan oleh event sebelumnya, hanya saja ada tambahan data **marker** pada parameter. Data tersebut berfungsi sebagai penanda bahwa sudah ada satu pemain yang telah menetapkan karakter yang akan dimainkan. Setelah *server* menerima event tersebut, maka *server* akan meneruskan data-data tersebut pada *host* dengan mengirimkan event **charSent(val,id,marker)**.

Pemain kedua pun akan melakukan hal yang sama untuk memilih dan menetapkan karakter yang akan dimainkan. Apabila kedua pemain telah menetapkan karakter yang akan dimainkan, *host* akan memancarkan event **charIsReady(playerData)**. Data yang dikirimkan merupakan suatu *object array* yang berisi data para pemain dan data karakter yang telah dipilih. Event tersebut berfungsi untuk memberi informasi kepada *server* untuk pindah kehalaman selanjutnya.

#### 4.1.3 Sequence Memulai Permainan



Gambar 4.3: Proses memulai permainan.

Pada tahap ini, *server* akan memancarkan *event startTheGame(playerData)*. Seperti yang dijelaskan pada Gambar 4.3, data yang dikirimkan merupakan data yang didapatkan dari *event* sebelumnya. *Event* tersebut akan diterima oleh seluruh *client* yang berada didalam *room* tertentu, yang kemudian akan memulai permainannya. *Host* akan mengeksekusi *method init()* dan **beginCountDown(3)** untuk memulai permainan. *Method init()* berfungsi untuk mulai menggambar lintasan lari di halaman saat ini yang akan menjadi tempat para karakter dimainkan. *Method beginCountDown(3)* berfungsi untuk melakukan hitungan mundur selama tiga detik.

Para pemain yang menerima *event startTheGame(playerData)* akan mengeksekusi *method showInstruction(3)*. *Method* tersebut berfungsi untuk menampilkan instruksi untuk memainkan permainan selama tiga detik. Apabila hitungan mundur telah selesai, maka permainan akan dimulai.

Permainan dilakukan dengan cara menekan tombol-tombol yang ada pada halaman *smartphone*. Apabila pemain menekan tombol, maka *event stepClicked(id)* akan dipancarkan. *Server* akan menerima *event* tersebut, yang kemudian akan meneruskannya dengan memancarkan *event moveThePlayer(id)* kepada *host*. Setelah *event* tersebut diterima oleh *host*, maka *host* akan menjalankan *method* tertentu. Apabila *Player1* yang menekan tombol, maka *method readyPlayerOne()* akan dieksekusi oleh *host*. Apabila *Player2*, maka **readyPlayerTwo()** akan dieksekusi. Kedua *Method* tersebut berfungsi untuk memindahkan posisi karakter milik pemain dari posisi semula hingga posisi tertentu. Permainan akan berakhir apabila ada karakter yang menyentuh garis

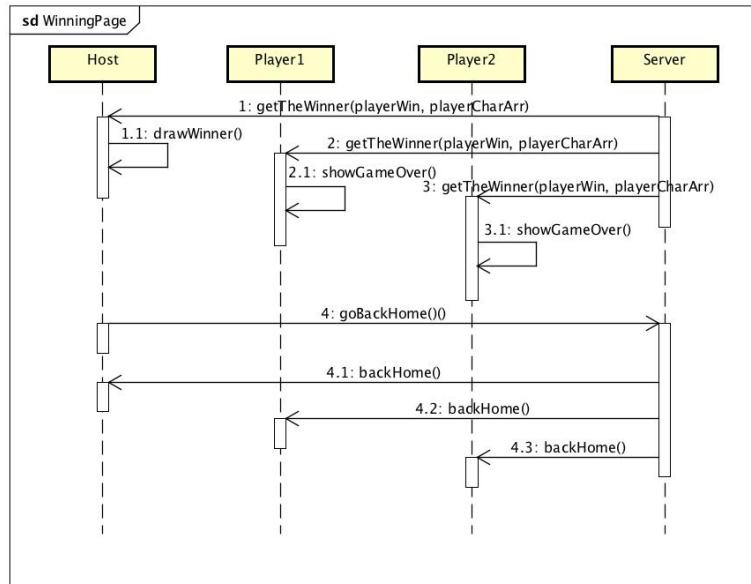
akhir pertama kali.

Apabila ada pemain yang berhasil menyentuh garis akhir, maka *host* akan memancarkan *event sendingTheWinner(playerWin, playerCharArr)*. Data-data yang dikirimkan akan dijelaskan sebagai berikut:

- **playerWin** angka *integer* yang menandakan pemain nomor berapa yang memenangkan permainan.
- **playerCharArr** *array* yang menyimpan karakter dari masing-masing pemain.

*Server* akan menangkap *event* tersebut dan meneruskannya dengan memancarkan *event toWinnerPage(playerWin, playerCharArr)* ke setiap *client* pada *room* tertentu. *Event* tersebut berfungsi untuk berpindah ke halaman selanjutnya.

#### 4.1.4 Sequence Mengakhiri Permainan



Gambar 4.4: Menampilkan para pemain yang telah selesai bermain.

Pada tahap ini, *server* akan memancarkan *event getTheWinner(playerWin, playerCharArr)* dengan data-data yang diterima dari *event* sebelumnya. Seperti yang dijelaskan pada Gambar 4.4 *host* akan menerima *event* tersebut dan akan mengeksekusi *method drawWinner()*. *Method* tersebut berfungsi untuk menampilkan karakter milik para pemenang yang telah selesai bermain. Para pemain akan menerima *event* yang dipancarkan oleh *server* dan mengeksekusi *method showGameOver()*. *Method* ini akan menampilkan teks yang menunjukkan bahwa permainan telah selesai.

Apabila pemain menekan tombol *back* yang ada pada halaman *browser*, maka *host* akan memancarkan *event goBackHome()*. *Event* tersebut akan diterima oleh *server* dan akan diteruskan dengan memancarkan *event backHome()*. Seluruh *client* didalam *room* yang menerima *event* tersebut akan kembali kehalaman semula dan koneksi pun akan terputus.

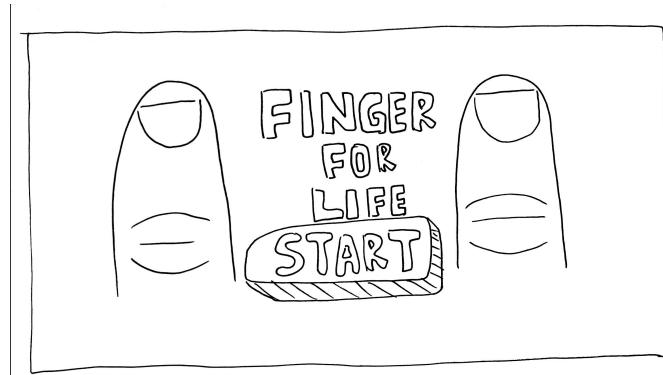
## 4.2 Perancangan Antarmuka

Antarmuka yang dirancang terbagi menjadi dua bagian, yaitu antarmuka pada *browser* yang ada di *PC* dan *smartphone*.

## 1. Antarmuka halaman *home*

### PC

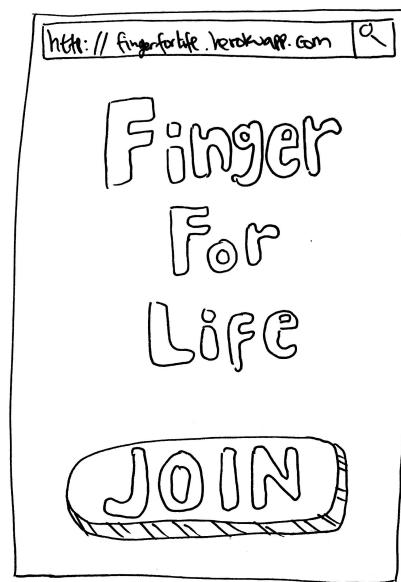
Halaman ini merupakan halaman utama yang pertama kali dituju oleh pengguna yang menggunakan *PC*. Komponen halaman ini terdiri dari dua buah gambar jari yang melambangkan permainan, teks yang menunjukkan nama permainan yaitu *finger for life*, dan tombol *start* yang digunakan untuk memulai permainan. Rancangan antarmuka halaman *home* dapat dilihat pada Gambar 4.5



Gambar 4.5: Halaman pada *PC* yang menunjukkan tampilan awal saat *client* mengakses alamat web.

### Smartphone

Halaman ini merupakan halaman utama yang pertama kali dituju oleh pengguna yang menggunakan *smartphone*. Komponen halaman ini terdiri dari teks yang menunjukkan nama permainan yaitu *finger for life*, dan tombol *join* yang digunakan untuk memulai permainan. Rancangan antarmuka halaman *home* dapat dilihat pada Gambar 4.6.

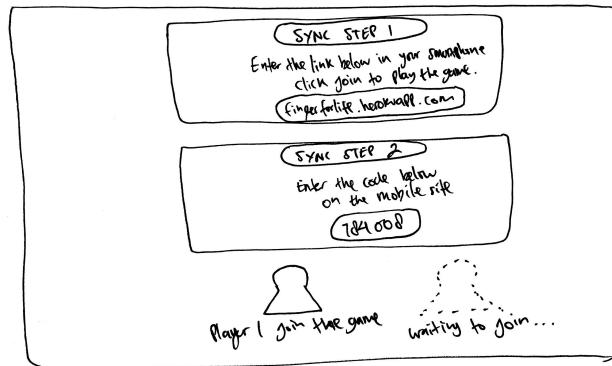


Gambar 4.6: Halaman pada *mobile* yang tampilan awal saat *client* mengakses alamat web.

## 2. Antarmuka halaman *sync*

### PC

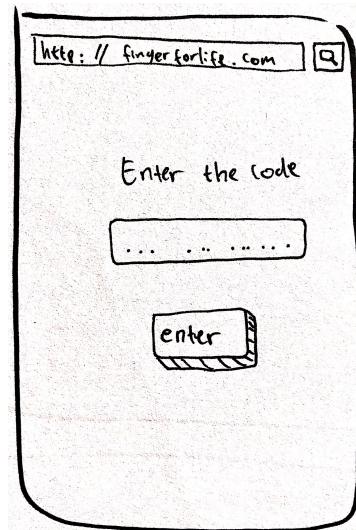
Halaman ini menampilkan suatu perintah yang dapat dilakukan oleh pengguna apabila akan bergabung dalam permainan. Halaman ini pun menyediakan suatu kode untuk para pemain dimana kode tersebut akan berfungsi sebagai suatu *room* bagi para pemain. Pada halaman ini akan dilakukan proses sinkronisasi untuk para pengguna, apakah berhasil bergabung dalam permainan atau tidak. Apabila berhasil, maka akan muncul suatu teks yang menunjukkan suatu pemain telah berhasil bergabung. Apabila tidak, maka tidak akan menampilkan apapun dan halaman tidak akan menuju ke halaman selanjutnya sebelum ada dua pemain yang berhasil bergabung. Rancangan antarmuka halaman *sync* pada *PC* dapat dilihat pada Gambar 4.7.



Gambar 4.7: Halaman pada *PC* yang menampilkan langkah untuk bergabung dalam permainan

### Smartphone

Halaman ini berfungsi untuk melakukan *request* untuk bergabung dalam permainan. Komponen halaman ini terdiri dari kolom kode, dan tombol *send*. Rancangan antarmuka halaman *sync* pada *smartphone* dapat dilihat pada Gambar 4.8.

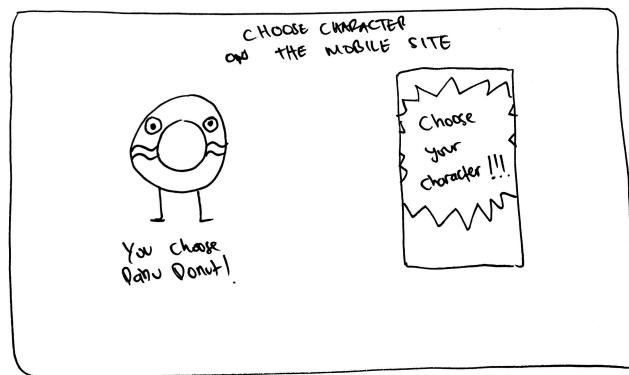


Gambar 4.8: Halaman pada *smartphone* yang menampilkan kolom untuk mengisi kode.

### 3. Antarmuka halaman *character*

#### PC

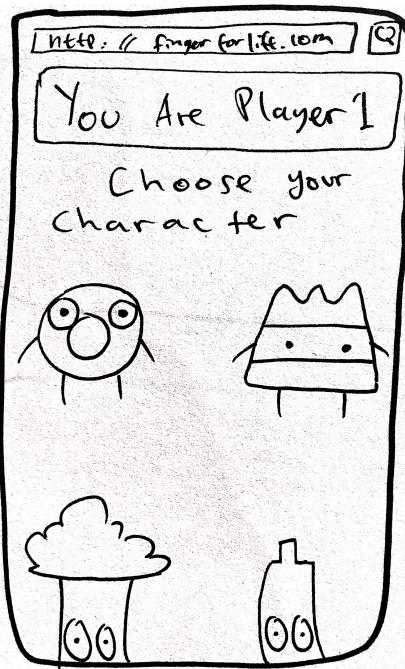
Halaman ini akan menampilkan karakter yang telah dipilih oleh pemain. Apabila karakter belum memilih karakter yang akan dimainkan, maka halaman ini belum menampilkan apapun. Rancangan antarmuka halaman *character* pada *PC* dapat dilihat pada Gambar 4.9.



Gambar 4.9: Halaman pada *PC* yang menampilkan karakter yang telah ditetapkan oleh pemain.

### Smartphone

Halaman ini akan menampilkan daftar karakter yang dapat dimainkan oleh pemain. Komponen halaman ini terdiri dari daftar karakter, dan tombol *choose*. Rancangan antarmuka halaman *character* pada *smartphone* dapat dilihat pada Gambar 4.10.

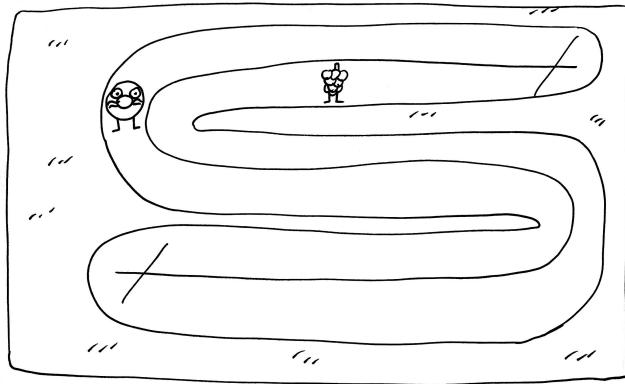


Gambar 4.10: Halaman pada *smartphone* yang menampilkan daftar karakter yang dapat dipilih.

#### 4. Antarmuka halaman *gameplay*

##### **PC**

Halaman ini menampilkan arena permainan untuk para pemain. Komponen halaman ini terdiri dari suatu gambar lintasan lari, dan karakter yang telah dipilih pada halaman sebelumnya. Rancangan antarmuka halaman *gameplay* pada *PC* dapat dilihat pada Gambar 4.11.



Gambar 4.11: Halaman pada *PC* yang menampilkan lintasan lari dan karakter untuk dimainkan.

Untuk menampilkan karakter yang telah dipilih kelayar *PC*, ada beberapa langkah yang harus dilakukan. Langkah-langkah tersebut akan dijelaskan sebagai berikut:

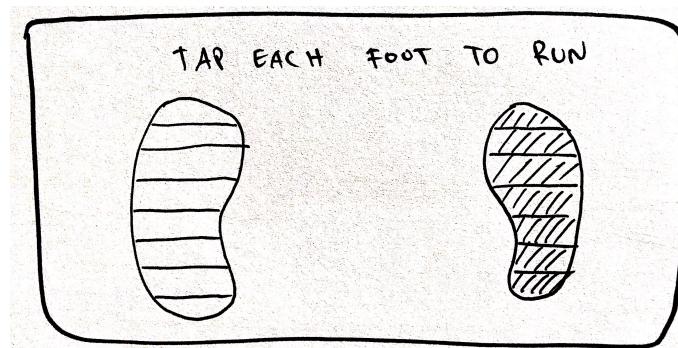
- *Client smartphone* akan mengirimkan nilai dari karakter tertentu yang dipilih. Nilai tersebut dikirimkan melalui *event Socket.io* yang dipancarkan dengan nama *sendingChar*. *Server* akan menangkap *event* tersebut, kemudian akan memancarkan *event startTheGame*, dimana *event* tersebut akan ditangkap oleh *host* untuk memulai permainan.
- *Host* akan mulai menggambar karakter pada *canvas* setelah mendapatkan data-data yang dikirimkan melalui *event startTheGame*.

Untuk menampilkan karakter kelayar *PC* saat proses animasi, ada beberapa langkah yang harus dilakukan. Langkah-langkah tersebut dijelaskan sebagai berikut:

- (a) *Canvas* akan menetapkan properti *globalCompositeOperation* dengan nilai *destination-over*. Nilai tersebut akan menyebabkan objek yang ditaruh pada *canvas* saat ini, akan berada dibelakang objek sebelumnya.
- (b) *State* dari *canvas* akan diubah ulang dengan cara menghilangkan seluruh objek yang berada didalam *canvas*. Proses tersebut akan dilakukan dengan menggunakan *method clearRect(0, 0, 900, 600)*.
- (c) Proses pengecekan akan dilakukan kepada seluruh pemain. Apabila pemain berada pada posisi x dan y tertentu, maka *state* dari *canvas* akan disimpan dengan menggunakan *save()*. Setelah itu, karakter milik kedua pemain akan digambar ke *canvas* dengan menggunakan *drawImage()*. Setelah proses tersebut selesai, maka seluruh *state* dari *canvas* akan dikembalikan ke *state* sebelumnya dengan menggunakan *restore()*.
- (d) Setelah proses pengecekan selesai, *state* dari *canvas* akan dikembalikan ke *state* sebelumnya dengan menggunakan *restore()*. Setelah itu akan dilakukan proses penggambaran lintasan lari dengan menggunakan *drawImage()*. Posisi lintasan lari yang digambar akan berada dibelakang gambar karakter yang sudah digambar sebelumnya.
- (e) Proses ini akan diulang kembali dari awal selama waktu permainan berjalan.

### Smartphone

Halaman ini berfungsi sebagai *controller* untuk para pemain. Komponen halaman ini terdiri dari dua buah gambar telapak kaki yang berfungsi sebagai tombol. Pemain dapat menekan tombol berulang kali untuk menggerakan karakter yang ada pada halaman *gameplay* di *PC*. Rancangan antarmuka halaman *gameplay* pada *smartphone* dapat dilihat pada Gambar 4.12.

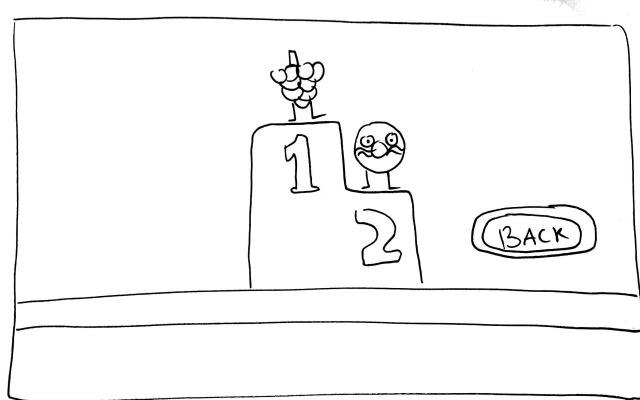


Gambar 4.12: Halaman pada *smartphone* yang menampilkan telapak kaki yang berfungsi sebagai pengendali.

##### 5. Antarmuka halaman *gameover*

###### PC

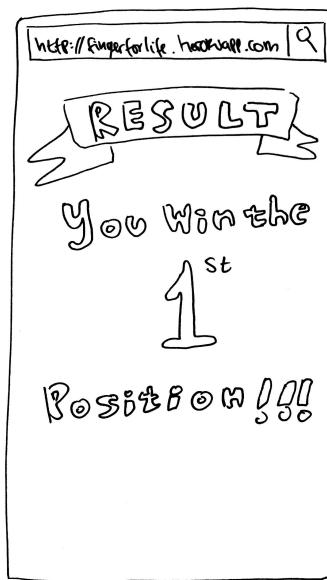
Halaman ini menampilkan para pemenang yang telah berhasil menyelesaikan permainan. Setelah pemain selesai bermain, pemain dapat menekan tombol *back* untuk kembali ke halaman utama dan mengakhiri permainan. Rancangan antarmuka halaman *gameover* pada *PC* dapat dilihat pada Gambar 4.13.



Gambar 4.13: Halaman pada *PC* yang menampilkan pemenang permainan.

###### Smartphone

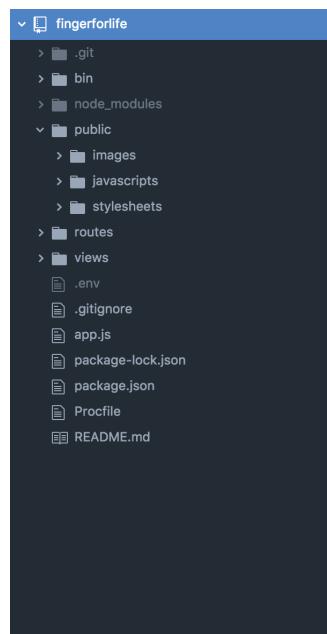
Halaman ini menampilkan teks yang menandakan bahwa permainan telah selesai. Rancangan antarmuka halaman *gameover* pada *smartphone* dapat dilihat pada Gambar 4.14.



Gambar 4.14: Halaman pada *smartphone* yang menampilkan teks bahwa permainan telah selesai.

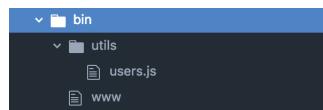
### 4.3 Perancangan Struktur Direktori

Bagian ini akan membahas mengenai perancangan struktur direktori untuk membangun web. Struktur direktori meliputi beberapa folder dan *file* yang memiliki fungsi berbeda-beda. Folder beserta *file* yang digunakan akan dijelaskan sebagai berikut:



Gambar 4.15: Direktori seluruh folder dan *file*.

#### 1. Folder bin



Gambar 4.16: Isi folder bin.

Folder ini berisi bagian-bagian yang berperan sebagai *server*. Pada folder ini terdapat folder lain dan berkas yang berfungsi untuk menangani jalannya koneksi pada saat web diakses. Berikut merupakan isi dari folder ini:

(a) Folder **utils**

Folder ini menyimpan berkas yang membantu proses berjalannya koneksi untuk *server*. Isi dari folder ini adalah sebagai berikut:

- Berkas **users.js**

Berkas ini merupakan suatu kelas yang berfungsi untuk menyimpan seluruh *client* yang terkoneksi pada *socket.io server*. Seluruh data pengguna yang akan memainkan permainan web akan disimpan dan diatur oleh berkas ini.

*Method-method* yang dimiliki oleh berkas ini akan dijelaskan sebagai berikut:

- **addUser(id, room)**

*Method* ini akan menambahkan pengguna kedalam *array*.

**Parameter:**

- \* **id**, identifikasi unik Socket.io milik pengguna.

- \* **room**, kode *room* milik pengguna.

**Kembalian:** Objek Users

- **IsRoomExist(room)**

*Method* ini akan memeriksa apakah parameter *room* cocok dengan *room* yang ada didalam *array*.

**Parameter:**

- \* **room**, kode *room* yang akan dicek.

- **Kembalian:** Integer

- **getUser(id)**

*Method* ini akan mendapatkan pengguna dengan parameter *id* yang cocok dengan *id* didalam *array*.

**Parameter:**

- \* **id**, identifikasi unik Socket.io milik pengguna.

**Kembalian:** Objek Users

- **removeUser(id)**

*Method* ini akan menghapus pengguna dengan *id* tertentu dari dalam *array*.

**Parameter:**

- \* **id**, identifikasi unik Socket.io milik pengguna.

**Kembalian:** Objek Users

- **getUserList(room)**

*Method* ini akan mengembalikan seluruh pengguna yang berada didalam *room* yang cocok dengan parameter *room*.

**Parameter:**

- \* **room**, kode *room* milik pengguna.

**Kembalian:** Array objek Users

(b) Berkas **www**

Berkas ini berfungsi sebagai *server*. Seluruh koneksi yang tersambung dan terputus akan diatur oleh berkas ini. Komunikasi antara *client* pun akan diatur oleh *server*.

Properti yang dimiliki oleh berkas ini akan dijelaskan sebagai berikut:

- **app**, modul app
- **debug**, modul debug
- **http**, modul http
- **socketIO**, modul Socket.io
- **Users**, modul Users
- **port**, nilai *port* yang akan dituju oleh *server*
- **server**, objek http *server*
- **io**, *server* Socket.io

*Method-method* yang dimiliki oleh berkas ini akan dijelaskan sebagai berikut:

- **normalizePort(val)**

*Method* ini berfungsi untuk mengubah bentuk parameter *val* menjadi *String*, *Integer*, atau *false*. // **Parameter**:

- **val**, parameter *port*

**Kembalian:** Objek *String*, *Integer*, atau *Boolean*.

*Event* Socket.io yang dimiliki berkas ini akan dijelaskan sebagai berikut:

- **io.on('connection', (socket) => ...)**

Berfungsi untuk menangani suatu koneksi Socket.io yang baru tersambung.

- **socket.on('hostJoinRoom', (msg) => ...)**

Berfungsi untuk menangkap *event* hostJoinRoom yang dipancarkan oleh *host* pada saat *host* akan masuk kedalam room.

- **socket.on('sendingTheWinner', (winner) => ...)**

Berfungsi untuk menangkap *event* sendingTheWinner yang dipancarkan oleh *client* setelah ada pemenang permainan.

- **socket.on('goBackHome', (msg) => ...)**

Berfungsi untuk menangkap *event* goBackHome yang dipancarkan oleh *client* saat permainan telah berakhir.

- **socket.on('roomFull', (msg) => ...)**

Berfungsi untuk menangkap *event* roomFull yang dipancarkan oleh *client* pada saat room telah terisi penuh.

- **socket.on('selectingChar', (msg) => ...)**

Berfungsi untuk menangkap *event* selectingChar yang dipancarkan oleh *client* pada saat karakter telah dipilih oleh pemain.

- **socket.on('sendingChar', (msg) => ...)**

Berfungsi untuk menangkap *event* sendingChar yang dipancarkan oleh *client* pada saat karakter telah ditetapkan oleh pemain.

- **socket.on('charIsReady', (msg) => ...)**

Berfungsi untuk menangkap *event* charIsReady yang dipancarkan oleh *client* pada saat kedua karakter telah ditetapkan.

- **socket.on('stepClicked', (message) => ...)**

Berfungsi untuk menangkap *event* stepClicked yang dipancarkan oleh *client* pada saat tombol telapak kaki ditekan oleh pemain.

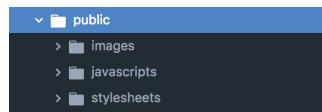
- **socket.on('requestToJoin', (msg) => ...)**

Berfungsi untuk menangkap *event* requestToJoin yang dipancarkan oleh *client* pada saat *client* melakukan proses permintaan bergabung.

- **socket.on('disconnect', () => ...)**

Berfungsi untuk menangkap *event* disconnect yang dipancarkan oleh *client* pada saat koneksi telah terputus.

## 2. Folder public



Gambar 4.17: Isi dari folder public.

Folder ini menyimpan bagian-bagian yang memiliki sifat *static*. Berikut akan dijelaskan isi dari folder ini:

### (a) Folder images

Folder ini berisi gambar-gambar yang dibutuhkan untuk ditampilkan di beberapa halaman yang terdapat didalam web. Berikut merupakan daftar isi dari gambar-gambar yang ada didalam folder ini:

- **BrocoDude.png** gambar karakter brokoli dengan ukuran besar.
- **brocoDudeTiny.png** gambar karakter brokoli dengan ukuran kecil.
- **DabuDonut.png** gambar karakter donat dengan ukuran besar.
- **dabuDonut.png** gambar karakter donat dengan ukuran kecil.
- **finga.png** gambar jari yang ditampilkan dihalaman awal.
- **GrapeYoda.png** gambar karakter anggur dengan ukuran besar.
- **grapeYodaTiny.png** gambar karakter anggur dengan ukuran kecil.
- **stepUp.png** gambar telapak kaki kiri yang ditampilkan dihalaman pengendali.
- **stepUpRight.png** gambar telapak kaki kanan yang ditampilkan dihalaman pengendali.
- **grapeYodaTiny.png** gambar karakter anggur dengan ukuran kecil.
- **SummerEgg.png** gambar karakter telur mata sapi dengan ukuran besar.
- **summerEggTiny.png** gambar karakter telur mata sapi dengan ukuran kecil.
- **timer1.png** gambar angka 1 yang ditampilkan saat hitungan mundur.
- **timer2.png** gambar angka 2 yang ditampilkan saat hitungan mundur.
- **timer3.png** gambar angka 3 yang ditampilkan saat hitungan mundur.
- **track.png** gambar lintasan saat permainan dimulai.
- **winning.png** gambar panggung saat sudah ada pemenang dalam permainan.

### (b) Folder javascripts

Folder ini berisi bagian-bagian yang berfungsi untuk mengatur bagaimana perilaku web pada saat diakses oleh pengguna. Terdapat berbagai berkas dengan ekstensi **.js** yang memiliki fungsi berbeda-beda. Berikut akan dijelaskan isi dari folder ini:

#### i. Folder libs

Folder ini berisi berkas yang berfungsi untuk mengatur pengiriman data melewati *form*. Berikut isi dari folder ini:

##### • **jquery-3.3.1.min.js**

Berkas ini merupakan **jquery** yang menyediakan fungsi-fungsi dalam pengiriman dan penerimaan data yang dilakukan *client* dan *server*.

#### ii. **charDesktopScript.js**

Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman pemilihan karakter di *PC browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **marker**, *integer* yang menandakan ada berapa pemain yang sudah menetapkan karakter.

- **playerData**, *array* yang menyimpan objek pengguna dengan nilai karakter yang dipilihnya.
- **imageArray**, *array* yang menyimpan daftar karakter yang dapat dipilih.

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **toGamePlayDesktop()**

Berfungsi untuk memindahkan halaman ke halaman permainan.

*Event* Socket.io yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket.on('charSelecting', function(msg){})**

Berfungsi untuk menangkap *event* charSelecting yang dipancarkan oleh *server* pada saat karakter yang dipilih akan ditampilkan ke layar *PC*.

- **socket.on('charSent', function(msg){})**

Berfungsi untuk menangkap *event* charSent yang dipancarkan oleh *server* untuk memeriksa apakah kedua pemain telah menetapkan karakter untuk pindah ke halaman selanjutnya.

- iii. **charMobileScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman pemilihan karakter di *mobile browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **mark**, *integer* yang menandakan ada berapa pemain yang sudah menetapkan karakter.

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **selectChar()**

Berfungsi untuk mengirimkan nilai karakter yang telah dipilih oleh pemain.

*Event* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket.on('charSent', function(msg))**

Berfungsi untuk menangkap *event* charSent yang dipancarkan oleh *server* untuk memeriksa apakah kedua pemain telah menetapkan karakter untuk pindah ke halaman selanjutnya.

- **socket.emit('selectingChar', val, id)**

Berfungsi untuk memancarkan *event* selectingChar saat pemain telah memilih karakter. Data yang dikirimkan adalah:

– **val**, nilai karakter berupa *integer*.

– **id**, identifikasi unik Socket.io milik pemain.

- **socket.on('sendingChar', val, id, marker)**

Berfungsi untuk memancarkan *event* sendingChar saat pemain telah menetapkan karakter yang dipilih. Data yang dikirimkan adalah:

– **val**, nilai karakter berupa *integer*.

– **id**, identifikasi unik Socket.io milik pemain.

– **marker**, *integer* penanda seorang pemain telah menetapkan karakter.

- iv. **gamePlayDesktopScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman permainan di *PC browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **canvas**, elemen *canvas* yang ada pada berkas HTML.

- **ctx**, menyimpan fungsi-fungsi konteks 2d milik *Canvas API* untuk pemain pertama.

- **ctx2**, menyimpan fungsi-fungsi konteks 2d milik *Canvas API* untuk pemain kedua.

- **track**, gambar lintasan lari.

- **timer1**, gambar angka 1 yang akan digunakan pada *countdown*.

- **timer2**, gambar angka 2 yang akan digunakan pada *countdown*.
- **timer3**, gambar angka 3 yang akan digunakan pada *countdown*.
- **player1Char**, gambar karakter milik pemain pertama.
- **player2Char**, gambar karakter milik pemain kedua.
- **dataOfPlayer**, menyimpan data kedua pemain.
- **player1Val**, menyimpan nilai karakter pemain pertama yang telah dipilih.
- **player2Val**, menyimpan nilai karakter pemain kedua yang telah dipilih.
- **winner**, menyimpan pemain pemenang.
- **timerArray**, menyimpan kumpulan gambar angka untuk proses *coundown*.
- **charArray**, menyimpan kumpulan gambar karakter.
- **arrOfPlayerChar**, menyimpan kedua karakter yang telah dipilih oleh pemain.
- **aniFrame**, menyimpan *animation frame* yang digunakan untuk proses animasi karakter pemain pertama.
- **aniFrame2**, menyimpan *animation frame* yang digunakan untuk proses animasi karakter pemain kedua.
- **progressPlayer1**, menyimpan kemajuan suatu posisi karakter yang digerakan pada lintasan lari milik pemain pertama.
- **progressPlayer2**, menyimpan kemajuan suatu posisi karakter yang digerakan pada lintasan lari milik pemain kedua.
- **xPosition1**, menyimpan posisi karakter pemain pertama pada sumbu x.
- **yPosition1**, menyimpan posisi karakter pemain pertama pada sumbu y.
- **xPosition2**, menyimpan posisi karakter pemain kedua pada sumbu x.
- **yPosition2**, menyimpan posisi karakter pemain kedua pada sumbu y.

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **init()**  
Berfungsi untuk menggambar lintasan lari diawal permainan.
- **beginCountDown(beginCounter, msg)**  
Berfungsi untuk memulai hitungan mundur dengan menggambar angka tiga hingga satu kelayar permainan.
- **drawChar(data)**  
Berfungsi untuk menggambar karakter milik pemain diatas lintasan lari.
- **beginGamePlay()**  
Berfungsi untuk memulai permainan.
- **reachFinishLine(player)**  
Berfungsi untuk mencatat pemain mana yang sampai pada garis akhir duluan.
- **toWinningPage()**  
Berfungsi untuk memindahkan halaman ke halaman pemenang.
- **readyPlayerOne()**  
Berfungsi untuk menggerakan karakter pemain pertama pada saat tombol telapak kaki ditekan.
- **readyPlayerTwo()**  
Berfungsi untuk menggerakan karakter pemain kedua pada saat tombol telapak kaki ditekan.

*Event* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket.on('moveThePlayer', function(msg)...)**  
Berfungsi untuk menangkap *event* moveThePlayer yang dipancarkan oleh *server* saat tombol telapak kaki ditekan oleh pemain.

- **socket.on('startTheGame', function(msg)...)**

Berfungsi untuk menangkap *event* startTheGame yang dipancarkan oleh *server* pada saat akan memulai permainan dengan menggambarkan asset-asset yang dibutuhkan ke layar *PC*.

- **socket.on('toWinnerPage', function())**

Berfungsi untuk menangkap *event* toWinnerPage yang dipancarkan oleh *server* yang memindahkan halaman ke halaman pemenang.

- **socket.emit('sendingTheWinner', playerWin,playerCharArr)**

Berfungsi untuk memancarkan *event* sendingTheWinner pada saat pemain telah mencapai garis akhir. Data yang dikirimkan adalah:

- **playerWin**, pemenang yang mencapai garis akhir lebih dulu.
- **playerCharArr**, nilai dari kedua karakter milik para pemain.

- gamePlayMobileScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman permainan di *mobile browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **stepLeftHtml**, Elemen HTML berupa gambar telapak kaki.
- **instructionEl**, Elemen HTML berupa teks instruksi.

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **stepClicked()**

Berfungsi untuk menangani aksi menekan tombol telapak kaki oleh pemain.

- **moveToWinPage()**

Berfungsi untuk memindahkan halaman ke halaman pemenang.

- **showInstruction(beginCounter)**

Berfungsi untuk menampilkan instruksi kelayar permainan.

**Parameter:**

- **beginCounter**, *integer* yang menunjukkan berapa detik instruksi akan ditampilkan.

*Event* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket.on('startTheGame', function(){...})**

Berfungsi untuk menangkap *event* startTheGame yang dipancarkan oleh *server* saat permainan akan segera dimulai.

- **socket.on('toWinnerPage', function(){...})**

Berfungsi untuk menangkap *event* toWinnerPage yang dipancarkan oleh *server* pada saat permainan telah selesai.

- **socket.emit('stepClicked',{playerId})**

Berfungsi untuk memancarkan *event* stepClicked pada saat pemain menekan tombol telapak kaki dilayar *smartphone*.

- homeScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman utama di *PC browser* dan *mobile browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **bg**, elemen HTML berupa *div* dengan nama kelas *stage-area*.

- **titleHtml**, elemen HTML berupa *div* dengan nama *id* *homePage*.

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **startClicked()**

Berfungsi untuk memindahkan halaman menuju halaman proses permintaan bergabung pada *PC*.

- **joinClicked()**

Berfungsi untuk memindahkan halaman menuju halaman proses permintaan bergabung pada *smartphone*.

vii. **mobileScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman proses permintaan koneksi di *mobile browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket**, variabel koneksi Socket.io

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **requestToJoin()**

Berfungsi untuk melakukan proses permintaan bergabung saat pengguna menekan tombol *send*.

- **toCharMobile()**

Berfungsi untuk memindahkan halaman ke halaman pemilihan karakter.

*Event* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket.on('connect',function(){}{})**

Berfungsi untuk menangkap *event connect* yang dipancarkan apabila *client* berhasil tersambung ke Socket.io.

- **socket.on('joinSucceed', function(msg){...})**

Berfungsi untuk menangkap *event joinSucceed* yang dipancarkan oleh *server* apabila *client* berhasil bergabung kedalam *room*.

- **socket.on('joinRejected', function(msg){...})**

Berfungsi untuk menangkap *event joinRejected* yang dipancarkan oleh *server* apabila *client* tidak berhasil bergabung kedalam *room*.

- **socket.on('toCharPage')**

Berfungsi untuk menangkap *event toCharPage* yang dipancarkan oleh *server* pada saat akan berpindah halaman ke halaman pemilihan karakter.

viii. **syncScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman proses permintaan koneksi di *PC browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket**, variabel koneksi Socket.io.

- **player**, *integer* yang menunjukan pemain urutan keberapa.

- **players**, *array* yang menyimpan data para pemain.

- **text**, *integer* kode *room*.

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **toCharDesk()**

Berfungsi untuk memindahkan halaman ke halaman pemilihan karakter pada *PC*.

- **getRandInt()**

Berfungsi untuk membangkitkan kode *room* berupa *integer* secara acak.

- **requestAccepted(msg)**

Berfungsi untuk memasukan data *client* kedalam *array* pada saat telah tersambung ke Socket.io.

*Event* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket.on('connect', function(){}{})**

Berfungsi untuk menangkap *event connect* pada saat *client* telah berhasil tersambung ke Socket.io.

- **socket.on('requestAccepted',function(msg){})**

Berfungsi untuk menangkap *event requestAccepted* yang dipancarkan oleh *server* pada saat *client* telah berhasil bergabung kedalam *room*.

- **socket.on('toCharPage', function(){}{})**

Berfungsi untuk menangkap *event toCharPage* yang dipancarkan oleh *server*

pada saat akan memindahkan halaman ke halaman pemilihan karakter pada *smartphone*

- **socket.emit('hostJoinRoom',{id, room})**

Berfungsi untuk memancarkan *event* hostJoinRoom saat *host* telah bergabung kedalam *room*. Data yang dikirimkan adalah:

- **id** identifikasi unik Socket.io milik *client*

- **room** kode *room* milik *client*

- **socket.emit('roomFull', {room})**

Berfungsi untuk memancarkan *event* hostJoinRoom pada saat *room* sudah terisi penuh.

- ix. **winningDesktopScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman saat permainan telah selesai di *PC browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **winCanvas**
- **winCtx**
- **podium**
- **winnerCharArr**

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **drawStage()**

Berfungsi untuk menggambar podium ke layar permainan.

- **drawWinner(winner)**

Berfungsi untuk menggambar para pemenang permainan.

- **drawFirstWinner(firstWinner)**

Berfungsi untuk menggambar pemenang pertama.

- **drawSecondWinner(secondWinner)**

Berfungsi untuk menggambar pemenang kedua.

- **toHomePage()**

Berfungsi untuk memindahkan halaman ke halaman awal.

*Event* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **socket.on('getTheWinner',function(msg){})**

Berfungsi untuk menangkap *event* getTheWinner yang dipancarkan oleh *server* pada saat permainan selesai.

- **socket.on('backHome', function(){})**

Berfungsi untuk menangkap *event* backHome yang dipancarkan oleh *server* untuk memindahkan halaman ke halaman awal.

- **socket.emit('goBackHome', 'back home')**

Berfungsi untuk memancarkan *event* goBackHome pada saat pemain menekan tombol *exit*.

- x. **winningMobileScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman saat permainan telah selesai di *mobile browser* pada saat diakses oleh *client*.

Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- **gameOverEl**, elemen HTML berupa teks.

*Method* yang dimiliki oleh berkas ini adalah sebagai berikut:

- **showGameOver()**

Berfungsi untuk menampilkan teks yang menandakan permainan sudah selesai.

- **backToHome()**

Berfungsi untuk memindahkan halaman ke halaman awal.

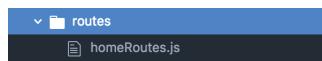
*Event* yang dimiliki oleh berkas ini adalah sebagai berikut:

(c) **Folder stylesheets**

Folder ini berisi bagian-bagian yang berfungsi untuk menghias setiap halaman yang terdapat didalam web. Pada folder ini terdapat berbagai berkas dengan ekstensi **.css**. Berikut akan dijelaskan isi dari folder ini:

- i. **charDesktopStyle.css**, Berkas ini berfungsi untuk menghias halaman pemilihan karakter pada *PC browser*.
- ii. **charMobileStyle.css**, Berkas ini berfungsi untuk menghias halaman pemilihan karakter pada *mobile browser*.
- iii. **gamePlayDesktopStyle.css**, Berkas ini berfungsi untuk menghias halaman permainan pada *PC browser*.
- iv. **gamePlayMobileStyle.css**, Berkas ini berfungsi untuk menghias halaman permainan pada *mobile browser*.
- v. **homeStyle.css**, Berkas ini berfungsi untuk menghias halaman utama pada *PC browser* dan *mobile browser*.
- vi. **mobileStyle.css**, Berkas ini berfungsi untuk menghias halaman proses permintaan koneksi pada *mobile browser*.
- vii. **Nickname\_DEMO.otf**<sup>1</sup>, Berkas ini berfungsi sebagai tipe *font* yang digunakan di beberapa halaman yang terdapat pada web.
- viii. **syncStyle.css**, Berkas ini berfungsi untuk menghias halaman proses permintaan koneksi pada *PC browser*.
- ix. **winningDesktopStyle.css**, Berkas ini berfungsi untuk menghias halaman saat permainan telah selesai pada *mobile browser*.
- x. **winningMobileStyle.css**, Berkas ini berfungsi untuk menghias halaman saat permainan telah selesai pada *mobile browser*.

### 3. Folder routes

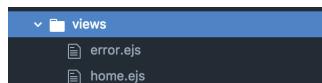


Gambar 4.18: Isi dari folder routes

Folder ini berisi bagian-bagian yang berperan sebagai *middleware*. Berikut merupakan isi dari folder ini:

- **homeRoutes.js**, berkas ini berfungsi sebagai *middleware* yang menampilkan halaman utama pada saat *client* mengakses alamat web. Berkas ini akan digunakan oleh berkas **app.js** yang berperan sebagai modul utama.

### 4. Folder views



Gambar 4.19: Isi dari folder views

Folder ini berisi bagian-bagian yang berfungsi untuk menampilkan halaman-halaman kepada *client*. Folder ini memiliki berkas dengan ekstensi **.ejs**. Sama seperti ekstensi **.html**, ekstensi

<sup>1</sup><http://pizzadude.dk/site/fonts/nickname>, diakses 12 September 2018

tersebut akan menampilkan berbagai tampilan dengan menggunakan *syntax .html*. Berikut merupakan isi dari folder ini:

- (a) **error.ejs**, berkas ini berisi *template* yang akan menampilkan halaman *error* apabila terjadi kesalahan dalam mengakses web. Halaman ini akan ditampilkan oleh *router*.
- (b) **home.ejs**, berkas ini berisi *template* yang akan menampilkan halaman awal pada saat pengguna mengakses web. Halaman ini akan ditampilkan oleh *router*.

#### 5. Berkas app.js

Berkas ini berfungsi sebagai modul utama, yang mengatur akses dari berbagai folder dan berkas yang ada pada struktur web ini.



## BAB 5

# IMPLEMENTASI DAN PENGUJIAN

Bab ini terdiri dari dua bagian, yaitu Implementasi Perangkat Lunak dan Pengujian Perangkat Lunak. Bagian implementasi berisi penjelasan lingkungan implementasi dan hasil implementasi. Sedangkan bagian pengujian berisi hasil pengujian fungsional dan eksperimental terhadap perangkat lunak yang telah dibangun.

### 5.1 Implementasi

#### 5.1.1 Lingkungan Implementasi

Lingkungan implementasi terbagi menjadi dua bagian, yaitu lingkungan perangkat keras dan lingkungan perangkat lunak. Kedua bagian tersebut akan dijelaskan sebagai berikut:

##### 1. Lingkungan Perangkat Keras

- (a) **Komputer:** Macbook Pro (Retina, 13 inch, Mid 2014)
- (b) **Prosesor:** 2.6 GHz Intel Core i5
- (c) **Memori:** 8 GB 1600 MHz DDR3
- (d) **Grafis:** Intel Iris 1536 MB

##### 2. Lingkungan Perangkat Lunak

Lingkungan perangkat lunak terbagi menjadi dua bagian yaitu, lingkungan pada *server* dan *client*. Kedua bagian tersebut akan dijelaskan sebagai berikut:

###### (a) Server

###### i. Heroku

Heroku merupakan *cloud platform* yang menyediakan fitur yang dapat membantu pengguna untuk dapat memiliki alamat domain. Spesifikasi Heroku yang digunakan oleh Finger For Life akan dijelaskan sebagai berikut:

Dyno Type	Sleeps	Professional Features	Memory (RAM)	CPU Share	Dedicated	Compute
free	yes	no	512 MB	1x	no	1x-4x

###### ii. Node.js

Node.js merupakan JavaScript *runtime* yang menyediakan fitur untuk mengeksekusi JavaScript dilingkungan lokal. Pengembangan Finger For Life menggunakan Node.js sebagai dasar penggunaan JavaScript dalam hal pengaturan *server* maupun *client*. Versi Node.js yang dipakai adalah 8.11.4.

###### iii. Socket.io

Socket.io merupakan teknologi yang menyediakan fitur komunikasi dua arah secara *real-time* antara *client* dan *server*. Pengembangan Finger For Life menggunakan

Socket.io sebagai cara komunikasi berdasarkan *event* antara *client* dan *server*. Dengan menggunakan Socket.io, maka proses komunikasi dalam permainan menjadi lebih efisien. Versi Socket.io yang dipakai adalah 2.0.4.

(b) **Client**

i. **Express.js**

Express.js adalah web *framework* untuk Node.js. *Framework* ini menyediakan kumpulan fitur yang dapat membantu dalam pembuatan aplikasi web. Pengembangan Finger For Life menggunakan Express.js untuk mengatur struktur direktori yang diperlukan. Dengan menggunakan Express.js, direktori dan berkas yang diperlukan untuk pengembangan dapat diatur peletakannya. Versi Express.js yang dipakai adalah 4.15.5.

ii. **Canvas API**

Pengembangan Finger For Life menggunakan Canvas API untuk membuat elemen-elemen grafis didalam web. Canvas API digunakan untuk membuat animasi permainan pada saat pemain menggerakan karakter yang ada dilintasan lari.

iii. **jQuery**

Pengembangan Finger For Life menggunakan Canvas API untuk mengatur akses elemen-elemen html yang ada didalam berkas-berkas tertentu. jQuery pun digunakan untuk mengirimkan data dalam bentuk form yang sudah diisi oleh pengguna. Versi jQuery yang dipakai adalah 3.3.1.

iv. **The Content Template element**

Pengembangan Finger For Life menggunakan The Content Template element untuk menampilkan kumpulan elemen HTML kelayar dengan menggunakan elemen <template>.

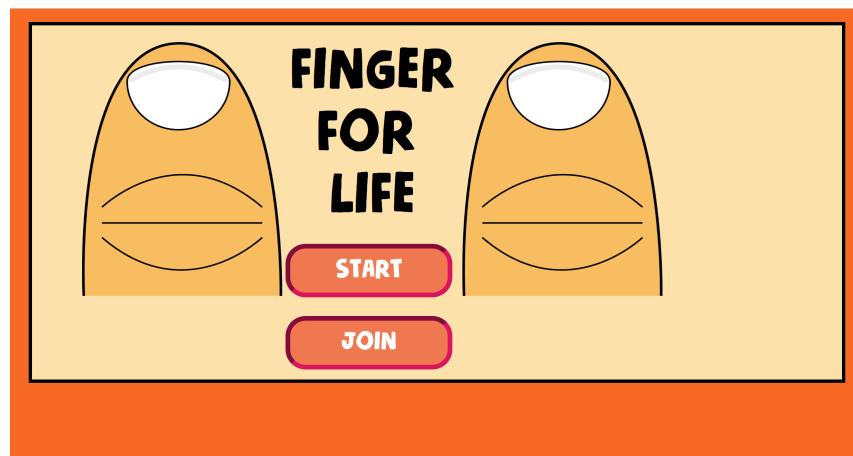
### 5.1.2 Hasil Implementasi

Hasil implementasi berupa aplikasi berbasis web yang berjalan berdasarkan Node.js. Aplikasi dapat diakses pada jaringan internet dengan URL <http://fingerforlife.herokuapp.com>. Aplikasi Finger For Life terdiri dari beberapa halaman. Halaman tersebut terbagi menjadi dua bagian, yaitu halaman pada *PC* dan pada *smartphone*. Halaman-halaman tersebut antara lain:

1. **Halaman *home***

**PC**

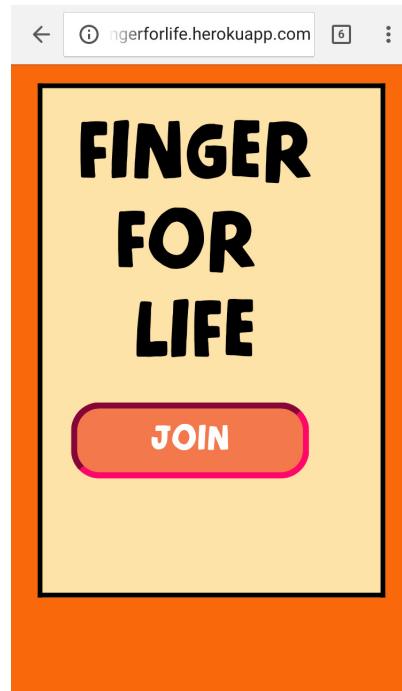
Halaman *home* pada *PC* digunakan pengguna untuk memulai permainan. Pada halaman ini terdapat tombol *start* yang berfungsi untuk meminta akses masuk kedalam permainan. Apabila pengguna menekan tombol tersebut, maka pengguna akan diarahkan kehalaman selanjutnya. Tangkapan layar dari halaman *home* pada *PC* dapat dilihat pada gambar 5.1.



Gambar 5.1: Halaman *Home*

### Smartphone

Halaman *home* pada *smartphone* digunakan pengguna untuk memulai permainan. Pada halaman ini terdapat tombol *join* yang berfungsi untuk meminta akses masuk kedalam permainan. Apabila pengguna menekan tombol tersebut, maka pengguna akan diarahkan kehalaman selanjutnya. Tangkapan layar dari halaman *home* pada *smartphone* dapat dilihat pada gambar 5.2.



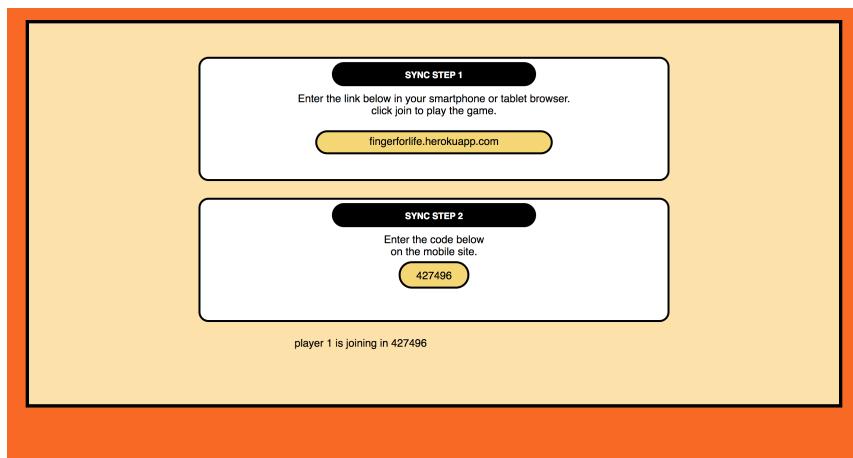
Gambar 5.2: Halaman *Home*

## 2. Halaman permintaan bergabung

### PC

Halaman *sync* pada *PC* berguna untuk proses sinkronisasi antara *PC* dan *smartphone*. Halaman ini menyediakan kode *room* yang harus dikirimkan oleh pengguna melalui *smartphone*. Apabila kode yang dikirimkan pengguna sesuai, maka pengguna akan bergabung kedalam *room* permainan. Apabila kode tidak sesuai, maka pengguna tidak bisa bergabung kedalam *room* permainan. Tangkapan layar dari halaman *sync* pada *PC* dapat dilihat pada gambar

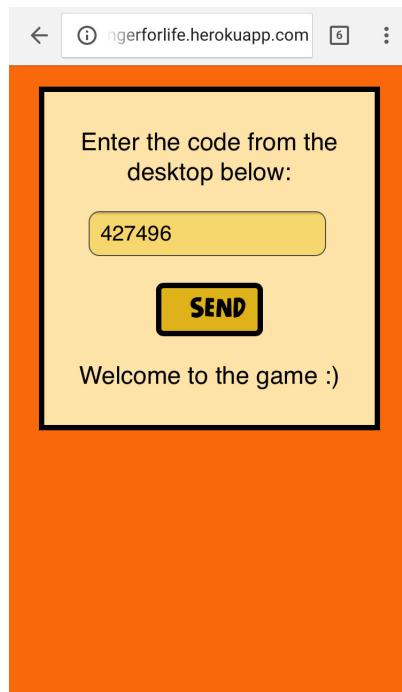
### 5.3.



Gambar 5.3: Halaman *Home*

#### Smartphone

Halaman *sync* pada *smartphone* berguna untuk proses sinkronisasi antara *PC* dan *smartphone*. Pengguna dapat mengisi kolom dengan kode yang didapatkan dari halaman web pada *PC*, kemudian menekan tombol *Send*. Apabila kode yang dikirimkan sesuai, maka akan muncul teks yang menunjukkan berhasil bergabung kedalam *room*. Apabila tidak sesuai, maka akan muncul teks yang menunjukkan tidak berhasil bergabung kedalam *room*. Tangkapan layar dari halaman *sync* pada *PC* dapat dilihat pada gambar 5.4.

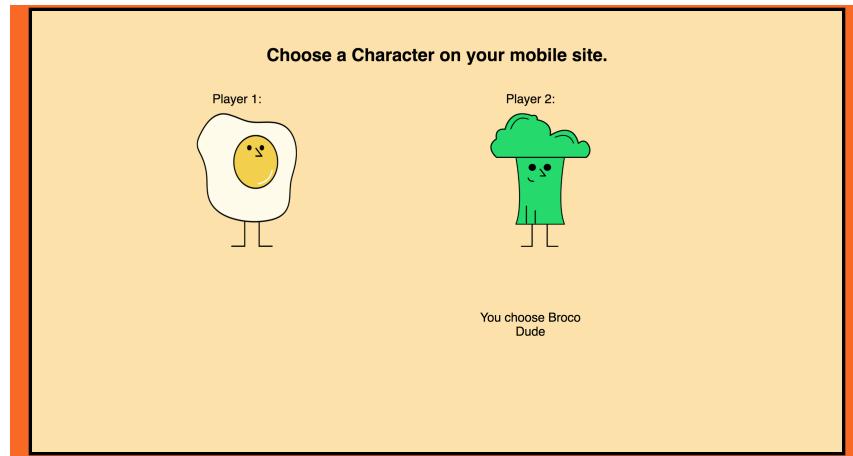


Gambar 5.4: Halaman *Home*

### 3. Halaman Pemilihan Karakter PC

Halaman pemilihan karakter pada *PC* akan menampilkan dua karakter yang telah dipilih oleh kedua pemain. Jika pemain menekan karakter yang terdapat pada halaman *smartphone*, maka

karakter tersebut akan ditampilkan. Jika pemain belum menekan karakter yang tersedia, maka halaman pada *PC* tidak akan menampilkan karakter. Tangkapan layar dari halaman pemilihan karakter pada *PC* dapat dilihat pada gambar 5.5.



Gambar 5.5: Halaman *Home*

### Smartphone

Halaman pemilihan karakter pada *smartphone* akan menampilkan daftar karakter permainan Finger For Life. Pemain dapat memilih satu karakter yang ditampilkan, kemudian menekan tombol *choose* untuk menetapkan karakter tersebut. Karakter yang telah ditetapkan akan dapat dimainkan saat permainan dimulai. Apabila kedua pemain telah menetapkan karakter, maka halaman akan berganti kehalaman selanjutnya. Tangkapan layar dari halaman pemilihan karakter pada *smartphone* dapat dilihat pada gambar 5.6.



Gambar 5.6: Halaman *Home*

## 4. Halaman Permainan *PC*

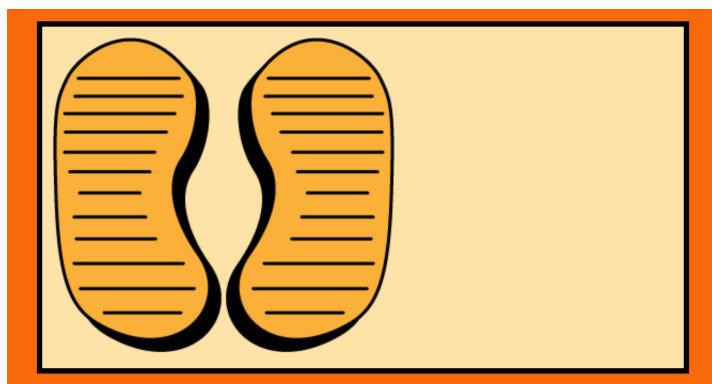
Halaman ini menampilkan lintasan lari dan kedua karakter yang dapat dimainkan. Karakter pada layar dapat bergerak melalui lintasan lari apabila pemain menekan tombol yang ada pada halaman *smartphone*. Tangkapan layar dari halaman *gameplay* pada *PC* dapat dilihat pada gambar 5.7.



Gambar 5.7: Halaman *Home*

### **Smartphone**

Halaman ini menampilkan tombol telapak kaki yang dapat ditekan oleh pemain. Tombol tersebut berfungsi untuk menggerakan karakter miliknya yang ditampilkan dihalaman *PC*. Pemain yang pertama kali menggerakan karakter hingga mencapai garis akhir menjadi pemenang, dan halaman akan berganti kehalaman selanjutnya. Tangkapan layar dari halaman *gameplay* pada *smartphone* dapat dilihat pada gambar 5.8.



Gambar 5.8: Halaman *Home*

## 5. Halaman Permainan Berakhir

### **PC**

Halaman ini menampilkan pemain yang memenangkan permainan. Pemenang pertama akan ditampilkan dipodium posisi atas, sedangkan pemenang kedua akan ditampilkan dipodium posisi bawah. Pemain dapat menekan tombol *exit* untuk keluar dari permainan. Tangkapan layar dari halaman *game over* pada *PC* dapat dilihat pada gambar 5.9.



Gambar 5.9: Halaman *Home*

### Smartphone

Halaman ini menampilkan teks yang menunjukkan posisi pemenang. Pemenang pertama akan ditampilkan teks yang menunjukkan posisi pertama, sedangkan pemenang kedua akan ditampilkan teks yang menunjukkan posisi kedua. Tangkapan layar dari halaman *game over* pada *smartphone* dapat dilihat pada gambar 5.10.



Gambar 5.10: Halaman *Home*

## 5.2 Pengujian

### 5.2.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui kesesuaian reaksi perangkat lunak dengan reaksi yang diharapkan berdasarkan aksi pengguna terhadap perangkat lunak. Pada saat pengujian fungsional dilakukan, penulis menemukan beberapa masalah yang muncul pada program aplikasi. Masalah tersebut muncul pada beberapa tahapan saat program dijalankan. Berdasarkan masalah-

masalah yang ditemukan, penulis juga mendapatkan solusi atas masalah yang ada. Berikut akan dijelaskan beberapa masalah yang ditemukan, beserta proses memecahkan masalah yang telah dilakukan.

### 1. Halaman *home*

- **Masalah:**

Tampilan home pada smartphone terkadang muncul gambar jari. Masalah ini ditemukan pada *smartphone* dengan resolusi layar yang berbeda-beda. Apabila ada *smartphone* dengan resolusi lebih besar dibanding yang lainnya, maka gambar jari muncul dilayar.

- **Solusi:**

Solusi dari masalah ini adalah dengan menggunakan fitur dari CSS yaitu *@media querry*. Fitur ini dapat mengatur pada resolusi berapakah suatu elemen ditampilkan kelayar. Berikut merupakan potongan kode solusi dari masalah ini.

```
@media (max-width:1260px){
    #left, #right, #startButton{
        display: none;
        float: none;
    }

    #joinButton{
        visibility: visible;
        width: 85%;
    }
}
```

Listing 5.1: Fitur CSS *@media querry*

### 2. Halaman permintaan bergabung

- **Masalah:**

Pada saat kedua pemain menyelesaikan permainan dan menekan tombol *exit*, kemudian kedua pemain akan mengulang permainan dengan menuju ke halaman permintaan bergabung. Pada tahap ini, salah satu pemain tidak dapat bergabung kedalam *room*.

*Error* ini terjadi akibat penggunaan elemen jQuery pada halaman permintaan bergabung, yaitu elemen *.submit()*. Pada saat pemain akan bermain kedua kalinya dan sampai dihalaman permintaan bergabung, elemen *submit()* akan mengirimkan kode room lebih dari satu kali. Dengan begitu, *server* akan mengira bahwa ada dua pemain yang sudah bergabung, sehingga halaman langsung berpindah kehalaman selanjutnya. Padahal pemain kedua belum mengisi kode *room* dan belum mengirimkan ke *server*.

- **Solusi:**

Solusi dari masalah ini adalah dengan menghilangkan elemen *submit()*, kemudian diganti dengan menggunakan Socket.io untuk memancarkan *event* pada saat tombol *send* ditekan. Berikut merupakan potongan kode dari solusi masalah ini.

```
function requestToJoin(){
    socket.emit('requestToJoin', {
        id: socket.id,
        room: $('#code').val()
    });
}
```

Listing 5.2: Proses memancarkan *event*

### 3. Halaman pemilihan karakter

- **Masalah:**

Pada saat salah satu pemain menekan tombol *choose* dua kali, halaman langsung pindah kehalaman *gameplay* disaat pemain kedua belum memilih karakter.

*Error* terjadi karena pada saat pemain menekan tombol *choose*, seluruh data langsung dikirimkan ke *server* sehingga *server* menangkap sudah ada satu pemain yang berhasil mengirimkan data karakter. Apabila ditekan dua kali, maka akan dianggap sudah ada dua pemain yang mengirimkan data karakter. Dengan begitu halaman pun berpindah.

- **Solusi:**

Solusi untuk masalah ini adalah dengan menggunakan JavaScript untuk menangani tombol *choose*, sehingga pemain tidak dapat mengirimkan data apabila belum memilih karakter. Setelah itu, tombol *choose* milik pemain yang telah memilih karakter akan dinonaktifkan, sehingga tidak dapat ditekan dua kali. Berikut merupakan potongan kode dari solusi masalah ini:

```
var valButton = $('input[name="radioChar"]:checked').val();
if (valButton === undefined) {
    alert("You have not choose a character.");
}
else{
    socket.emit('sendingChar', {
        val: valButton,
        id: socket.id,
        marker: 1
    });
    document.getElementById("nextButton").disabled = true;
}
```

Listing 5.3: Proses menangani tombol *choose*

### 4. Halaman pemilihan karakter

- **Masalah:**

Pada saat memilih karakter, setelah ditekan karakternya pada *smartphone*, karakter tidak muncul dilayar *PC*.

*Error* ini terjadi karena adanya kesalahan pengiriman data pada saat proses bergabung kedalam *room*. Yang masuk kedalam *room* hanya satu pemain saja, sedangkan pemain kedua tidak mengirimkan data kepada *server* sehingga tidak masuk kedalam *room*.

- **Solusi:**

Solusi untuk masalah ini adalah dengan menggunakan JavaScript untuk menangani tombol *choose*, sehingga pemain tidak dapat mengirimkan data apabila belum memilih karakter. Setelah itu, tombol *choose* milik pemain yang telah memilih karakter akan dinonaktifkan, sehingga tidak dapat ditekan dua kali. Berikut merupakan potongan kode dari solusi masalah ini:

```
function requestToJoin(){
    socket.emit('requestToJoin', {
        id: socket.id,
        room: $('#code').val()
    });
}
```

}

Listing 5.4: Proses menangani tombol *choose*

Pengujian fungsional terbagi menjadi dua, yaitu pengujian fungsional pada *PC* dan pada *smartphone*.

**PC**

No	Aksi Pengguna	Reaksi yang diharapkan	Reaksi perangkat lunak
1	Pengguna menjalankan aplikasi	Halaman <i>home</i> akan ditampilkan	sesuai
2	Pengguna menekan tombol <i>start</i>	Halaman akan diarahkan menuju halaman pemintaan bergabung	sesuai
3	Pengguna melakukan proses sinkronisasi <i>PC</i> dan <i>smartphone</i> .	Jika pemain pertama berhasil bergabung kedalam <i>room</i> , maka pada layar <i>PC</i> akan muncul pesan "Player 1 has join the room"	sesuai
		Jika pemain kedua berhasil bergabung ke-dalam <i>room</i> , maka pada layar <i>PC</i> akan muncul pesan "Player 2 has join the room", kemudian halaman langsung diarahkan menuju halaman pemilihan karakter	sesuai
4	Pengguna melakukan proses pemilihan karakter	Jika para pemain memilih karakter, maka kedua karakter yang dipilih akan ditampilkan kelayar <i>PC</i>	sesuai
		Jika para pemain telah menetapkan karakter yang dipilih, maka halaman akan diarahkan menuju halaman permainan	sesuai
5	Pengguna mulai memainkan permainan	Karakter milik pemain bergerak melalui lintasan lari selama permainan dan akan diarahkan kehalaman permainan berakhir apabila ada yang menyentuh garis akhir lebih dulu	sesuai
6	Pengguna menekan tombol <i>exit</i>	Permainan berakhir dan halaman diarahkan kembali menuju halaman <i>home</i>	sesuai

**Smartphone**

No.	Aksi Pengguna	Reaksi yang diharapkan	Reaksi perangkat lunak
1	Pengguna menjalankan aplikasi	Halaman <i>home</i> ditampilkan	sesuai
2	Pengguna menekan tombol <i>join</i>	Halaman akan diarahkan menuju halaman permintaan bergabung	sesuai
3	Pengguna memasukan kode <i>room</i> dan menekan tombol <i>send</i>	Jika kode <i>room</i> sesuai, maka akan muncul pesan "Welcome to the game !"	sesuai
		Jika kedua pemain berhasil bergabung, maka halaman akan diarahkan menuju halaman pemilihan karakter	sesuai
4	Pengguna menekan karakter pada layar <i>smartphone</i>	Karakter yang ditekan akan ditampilkan dilayar <i>PC</i>	sesuai
		Jika kedua pemain telah memilih karakter, maka halaman akan diarahkan menuju halaman permainan	sesuai
5	Pengguna menekan tombol telapak kaki secara berulang selama permainan	Karakter pada layar <i>PC</i> akan bergerak	sesuai
		Jika ada pemain yang lebih dulu mencapai garis akhir, maka halaman akan diarahkan menuju halaman permainan berakhir	sesuai
6	Pengguna mengakhiri permainan	Halaman diarahkan kembali menuju halaman <i>home</i>	sesuai

### 5.2.2 Pengujian Eksperimental

1. Pengujian *Cross-Platform*
2. Pengujian *latency*



## DAFTAR REFERENSI

- [1] Rauch, G. (2011) Socket.io. <https://socket.io/>. 3 September 2018.
- [2] Mozilla (2004) Canvas api. [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API). 3 September 2018.
- [3] Dahl, R. (2009) Node.js docs. <https://nodejs.org/dist/latest-v8.x/docs/api/>. 3 September 2018.
- [4] Holowaychuk, T. (2010) Express.js. <https://expressjs.com/>. 3 September 2018.
- [5] Dahl, R. (2009) Node.js. <https://nodejs.org/en/docs/guides/>. 3 September 2018.
- [6] Holowaychuk, T. (2010) Express.js documentation. <https://expressjs.com/en/4x/api.html>. 3 September 2018.
- [7] Rauch, G. (2011) Socket.io server api. <https://socket.io/docs/server-api/>. 3 September 2018.
- [8] Rauch, G. (2011) Socket.io client api. <https://socket.io/docs/client-api/>. 3 September 2018.
- [9] MIT (2006) "jquery". <https://jquery.com/>. 3 September 2018.
- [10] Mozilla (2006) jquery api. <https://api.jquery.com/>. 3 September 2018.
- [11] Mozilla (2005) The content template element. <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template>. 3 September 2018.



# LAMPIRAN A

## KODE PROGRAM

Listing A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[1] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = ( -aaa + &daa ) / ( bbb++ - ccc % 2 );
14             strcpy(a,"hello_@?");
15         }
16     count = ~mask | 0x00FF00AA;
17 }
18
19 // Fonts for Displaying Program Code in LATEX
20 // Adrian P. Robson, nepswb.co.uk
21 // 8 October 2012
22 // http://nepswb.co.uk/docs/progfonts.pdf
23

```

Listing A.2: MyCode.java

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the furthest edge
9     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID;           //store the ID of all vertices
12    protected ArrayList<Double> closeDist;          //store the distance of all vertices
13    protected int totaltrj;                          //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i < totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35}
36

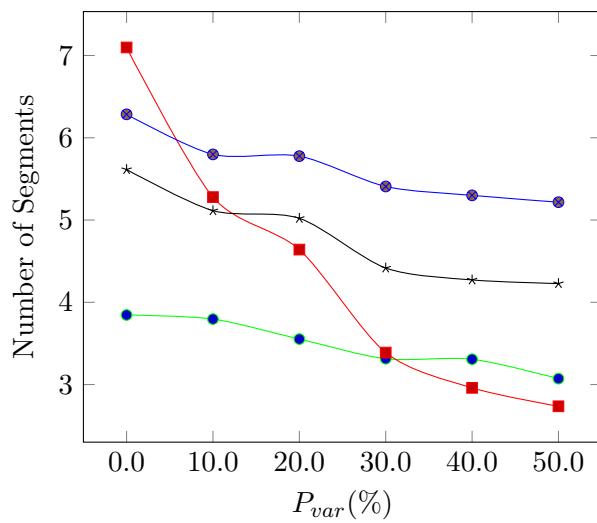
```



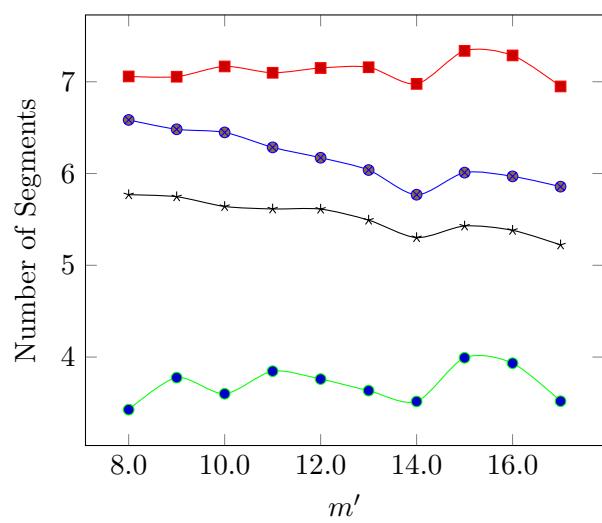
## LAMPIRAN B

### HASIL EKSPERIMENT

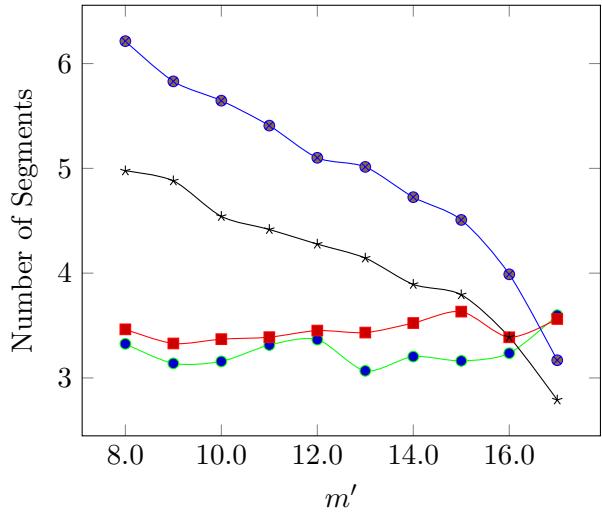
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



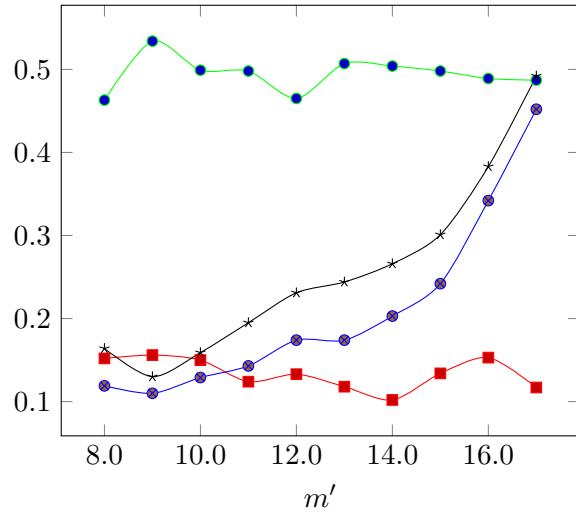
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4