

## SKRIPSI

### PEMANFAATAN SMARTPHONE SEBAGAI PENGENDALI PERMAINAN BERBASIS WEB



Priambodo Pangestu

NPM: 2013730055

PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS  
UNIVERSITAS KATOLIK PARAHYANGAN  
2018



**UNDERGRADUATE THESIS**

**UTILIZATION OF SMARTPHONE AS WEB-BASED GAME  
CONTROLLERS**



**Priambodo Pangestu**

**NPM: 2013730055**

**DEPARTMENT OF INFORMATICS  
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES  
PARAHYANGAN CATHOLIC UNIVERSITY  
2018**



## ABSTRAK

Socket.io merupakan sebuah pustaka yang menyediakan fitur untuk melakukan komunikasi secara *real-time* dan dua arah antara *browser* dan *server*. Dengan menggunakan Socket.io, *client* dapat mengirimkan pesan kepada *server* dan menerima respon tanpa harus melakukan *polling*, yang berarti proses pengecekan secara berulang terhadap *server* untuk mengetahui apakah *server* masih tersambung atau tidak. Fitur-fitur yang dimiliki oleh Socket.io dapat dimanfaatkan untuk mengembangkan aplikasi web yang membutuhkan komunikasi *real-time*. Salah satu pemanfaatan pustaka Socket.io adalah permainan berbasis web.

Permainan berbasis web yang akan dibangun dinamakan Finger For Life. Permainan ini memanfaatkan teknologi *smartphone* dan *PC*, dimana *smartphone* akan berperan sebagai pengendali didalam permainan, dan *PC* akan berperan sebagai *console* yang akan menyediakan permainan. Untuk memainkan permainan, *smartphone* harus terkoneksi ke *PC* melalui *browser*. Oleh karena itu, fitur yang dimiliki oleh Socket.io digunakan untuk melakukan koneksi antara *smartphone* dan *PC*.

Dengan penggunaan Socket.io didalam pengembangan aplikasi Finger For Life, diharapkan jumlah *latency* yang dihasilkan pada saat memainkan permainan akan sangat kecil. *Latency* merupakan jarak waktu yang dihasilkan pada saat suatu konten atau data dikirimkan dari *client* menuju *server*, maupun sebaliknya. Jumlah *latency* yang dihasilkan akan sangat berpengaruh pada saat tombol yang ada di *smartphone* ditekan, dengan respon yang diberikan oleh *PC* berdasarkan aksi tersebut. Semakin kecil jumlah *latency* yang dihasilkan, maka akan semakin cepat respon yang diberikan.

**Kata-kata kunci:** Socket.io, pemanfaatan, *smartphone*, *PC*, pengendali, permainan, web, *browser*, *latency*



## ABSTRACT

Socket.io is a library that enables real-time, bidirectional communication between the browser and the server. With this API, client can send messages to the server and receive responses without having to poll the server, which means check continuously to the server to see whether the server still connected or not. With these features, Socket.io can be used to build a real-time communication web application such as games.

The web-based games which utilize the features of Socket.io is Finger For Life. This game use smartphone as the controller and PC as the console. To be able to play, users need to connect the smartphone to the PC through web browser. Therefore, Socket.io is used to connect a smartphone to the PC.

The use of Socket.io in the developing Finger For Life web application is expected to decrease the sum of latency of playing the web-based game. Latency is a time interval when the content or the data is being sent from client to the server and back. The sum of latency can affect how fast the response from the PC when the users click the button on the smartphone. The lower the latency, the faster the response can be sent.

**Keywords:** Socket.io, utilization, smartphone, PC, controller, game, web, browser, latency



## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>ix</b>
<b>DAFTAR GAMBAR</b>	<b>xi</b>
<b>DAFTAR TABEL</b>	<b>xiii</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 . . . . .	3
1.5 Metodologi . . . . .	3
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 . . . . .	7
2.2.1 express() . . . . .	8
2.2.2 Application . . . . .	9
2.2.3 Response . . . . .	10
2.2.4 Router . . . . .	11
2.3 Socket.io . . . . .	12
2.3.1 Server API . . . . .	12
2.3.2 Client API . . . . .	14
2.4 Canvas API . . . . .	16
2.4.1 Animation . . . . .	16
2.4.2 canvasRenderingContext2D . . . . .	18
2.5 jQuery . . . . .	19
2.5.1 .submit(handler) . . . . .	19
2.5.2 .val() . . . . .	20
2.5.3 .html() . . . . .	20
2.5.4 .preventDefault() . . . . .	20
2.6 The Content Template element . . . . .	20
<b>3 ANALISIS</b>	<b>23</b>
3.1 Analisis Aplikasi Sejenis . . . . .	23
3.2 Analisis Alur Permainan Finger For Life . . . . .	29
3.3 Analisis Pengembangan Web . . . . .	31
3.4 Analisis <i>Use Case</i> . . . . .	43
3.4.1 Diagram <i>Use Case</i> . . . . .	43

3.4.2 Skenario <i>Use Case</i> . . . . .	43
3.5 Analisis Arsitektur Finger For Life . . . . .	45
3.6 Analisis Socket.io . . . . .	46
<b>4 PERANCANGAN</b>	<b>51</b>
4.1 Perancangan <i>Sequence Diagram</i> . . . . .	51
4.1.1 <i>Sequence</i> Permintaan Bergabung . . . . .	51
4.1.2 <i>Sequence</i> Memilih Karakter . . . . .	53
4.1.3 <i>Sequence</i> Memulai Permainan . . . . .	54
4.1.4 <i>Sequence</i> Mengakhiri Permainan . . . . .	56
4.2 Perancangan Antarmuka . . . . .	57
4.3 Perancangan Struktur Direktori . . . . .	63
<b>5 IMPLEMENTASI DAN PENGUJIAN</b>	<b>77</b>
5.1 Implementasi . . . . .	77
5.1.1 Lingkungan Implementasi . . . . .	77
5.1.2 Hasil Implementasi . . . . .	79
5.2 Pengujian . . . . .	85
5.2.1 Pengujian Fungsional . . . . .	85
5.2.2 Pengujian Eksperimental . . . . .	89
<b>6 KESIMPULAN DAN SARAN</b>	<b>95</b>
6.1 Kesimpulan . . . . .	95
6.2 Saran . . . . .	95
<b>DAFTAR REFERENSI</b>	<b>97</b>
<b>A KODE PROGRAM CLIENT</b>	<b>99</b>
A.1 Kode Program Direktori <i>public</i> . . . . .	99
A.1.1 Kode Program Halaman Utama . . . . .	99
A.1.2 Kode Program Halaman Permintaan Bergabung . . . . .	102
A.1.3 Kode Program Halaman Memilih Karakter . . . . .	106
A.1.4 Kode Program Halaman Memulai Permainan . . . . .	110
A.1.5 Kode Program Halaman Mengakhiri Permainan . . . . .	115
A.2 Kode Program Direktori <i>routes</i> . . . . .	116
A.2.1 Kode Program <i>homeRoutes.js</i> . . . . .	116
A.3 Kode Program Direktori <i>views</i> . . . . .	116
A.3.1 Kode Program Halaman <i>Error</i> . . . . .	116
A.3.2 Kode Program Seluruh Halaman pada Finger For Life . . . . .	116
A.4 Kode Program <i>app.js</i> . . . . .	119
<b>B KODE PROGRAM Server</b>	<b>121</b>
B.1 Kode Program Kelas <i>users.js</i> . . . . .	121
B.2 Kode Program <i>www</i> . . . . .	121

## DAFTAR GAMBAR

3.1 Halaman awal web AirConsole pada <i>PC</i> . . . . .	23
3.2 Kode yang harus dimasukan oleh pemain pada <i>smartphone</i> . . . . .	24
3.3 Halaman awal Airconsole pada <i>smartphone</i> . . . . .	24
3.4 Pemain diminta untuk memasukan kode yang sudah didapatkan pada <i>PC</i> . . . . .	25
3.5 Memasukan kode yang sudah didapatkan pada <i>PC</i> . . . . .	25
3.6 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih. . . . .	26
3.7 Halaman pada <i>smartphone</i> yang berfungsi sebagai pengendali. . . . .	26
3.8 Halaman awal permainan The Neighborhood pada <i>PC</i> . . . . .	27
3.9 Halaman awal permainan The Neighborhood pada <i>smartphone</i> . . . . .	27
3.10 Halaman pada <i>PC</i> dimana permainan sedang berlangsung. . . . .	27
3.11 Halaman pada <i>smartphone</i> dimana permainan sedang berlangsung. . . . .	28
3.12 Halaman pada <i>PC</i> apabila permainan sudah dimenangkan. . . . .	28
3.13 Halaman pada <i>smartphone</i> apabila permainan sudah dimenangkan. . . . .	28
3.14 Halaman pada <i>PC</i> yang menunjukan pemutusan koneksi. . . . .	29
3.15 Efek <i>destination-over</i> . . . . .	40
3.16 Diagram <i>use case</i> pemain . . . . .	43
3.17 Arsitektur Finger For Life . . . . .	45
3.18 Arsitektur interaksi <i>client</i> dan <i>server</i> . . . . .	46
4.1 Proses melakukan koneksi ke Socket.io dan bergabung kedalam <i>room</i> . . . . .	51
4.2 Proses memilih karakter. . . . .	53
4.3 Proses memulai permainan. . . . .	54
4.4 Menampilkan para pemain yang telah selesai bermain. . . . .	56
4.5 Halaman pada <i>PC</i> yang menunjukan halaman utama saat <i>client</i> mengakses alamat web. . . . .	57
4.6 Halaman pada <i>smartphone</i> yang menunjukan halaman utama saat <i>client</i> mengakses alamat web. . . . .	58
4.7 Halaman pada <i>PC</i> yang menampilkan langkah untuk bergabung kedalam permainan. . . . .	58
4.8 Halaman pada <i>smartphone</i> yang menampilkan kolom untuk mengisi kode. . . . .	59
4.9 Halaman pada <i>PC</i> yang menampilkan karakter yang telah ditetapkan oleh pemain. . . . .	59
4.10 Halaman pada <i>smartphone</i> yang menampilkan daftar karakter yang dapat dipilih. . . . .	60
4.11 Halaman pada <i>PC</i> yang menampilkan lintasan lari dan karakter untuk dimainkan. . . . .	60
4.12 Halaman pada <i>smartphone</i> yang menampilkan telapak kaki yang berfungsi sebagai pengendali. . . . .	62
4.13 Halaman pada <i>PC</i> yang menampilkan pemenang permainan. . . . .	62
4.14 Halaman pada <i>smartphone</i> yang menampilkan teks bahwa permainan telah selesai. . . . .	63
4.15 Seluruh direktori dan berkas yang digunakan didalam pengembangan aplikasi Finger For Life . . . . .	63
4.16 Isi direktori bin. . . . .	64
4.17 Isi dari folder public. . . . .	67
4.18 Isi dari folder routes . . . . .	75
4.19 Isi dari folder views . . . . .	75

5.1	Halaman utama pada <i>PC</i>	79
5.2	Halaman utama pada <i>smartphone</i>	80
5.3	Halaman permintaan bergabung pada <i>PC</i>	80
5.4	Halaman permintaan bergabung pada <i>smartphone</i>	81
5.5	Halaman memilih karakter pada <i>PC</i>	81
5.6	Halaman memilih karakter pada <i>smartphone</i>	82
5.7	Halaman memulai permainan pada <i>PC</i>	83
5.8	Halaman memulai permainan pada <i>smartphone</i>	83
5.9	Halaman mengakhiri permainan pada <i>PC</i>	84
5.10	Halaman mengakhiri permainan pada <i>smartphone</i>	84

## **DAFTAR TABEL**

5.1 Tabel Pengujian Fungsional pada <i>PC</i> . . . . .	88
5.2 Tabel Pengujian Fungsional pada <i>smartphone</i> . . . . .	89



1

## BAB 1

2

### PENDAHULUAN

#### 3 1.1 Latar Belakang

4 Socket.io adalah teknologi yang memungkinkan *client* dan *server* untuk melakukan komunikasi dua  
5 arah secara *real-time*, yang berarti memiliki selisih waktu yang sedikit antara *server* dan *client* [1].  
6 Socket.io memiliki dua bagian: *client-side library*, atau pustaka pada bagian *client* yang berjalan  
7 didalam *web browser*, dan *server-side library*, atau pustaka pada bagian *server* yang berjalan pada  
8 bagian *server*. Socket.io memiliki fitur untuk melakukan komunikasi dari satu *server* ke beberapa  
9 *client* didalam proses implementasinya. Teknologi ini sangat berguna untuk membantu membangun  
10 sebuah aplikasi yang membutuhkan koneksi *real-time* seperti dalam aplikasi permainan berbasis  
11 web.

12 Untuk memanfaatkan teknologi Socket.io dalam membangun aplikasi permainan, akan dibu-  
13 tuhkan beberapa teknologi yang dapat membantu pembangunan aplikasinya. Salah satu teknologi  
14 tersebut adalah Canvas API. Teknologi ini merupakan bagian dari elemen HTML5 yang dapat  
15 digunakan untuk mengolah objek grafis dengan menggunakan JavaScript [2]. *Canvas API* dapat  
16 juga digunakan untuk mengolah komposisi foto dan membuat animasi. Oleh karena itu, fungsi-fungsi  
17 yang ada pada *Canvas API* akan membantu pembangunan aplikasi permainan terutama pada  
18 bagian pengembangan grafis.

19 Teknologi lain yang dapat membantu membangun aplikasi permainan dalam menggunakan  
20 teknologi Socket.io adalah Node.js. Teknologi ini merupakan sebuah *platform* atau lingkungan yang  
21 didesain untuk mengembangkan aplikasi berbasis web pada bagian *server* [3]. Node.js ditulis dalam  
22 sintaks bahasa pemrograman JavaScript dan menggunakan *V8* yang merupakan *engine* JavaScript  
23 milik perusahaan *Google* untuk mengeksekusi JavaScript pada *web server*. Node.js memiliki sifat  
24 *non-blocking*, yang berarti Node.js tidak akan menunggu untuk mengerjakan permintaan selanjutnya.  
25 Oleh karena itu, Fitur-fitur yang dimiliki oleh Node.js akan sangat membantu untuk membangun  
26 aplikasi permainan yang membutuhkan koneksi *real-time*.

27 Salah satu teknologi yang akan membantu dalam mengimplementasi Node.js adalah Express.js  
28 [4]. Teknologi ini menyediakan kumpulan fitur untuk mengatur penyimpanan data lokal dalam  
29 membangun aplikasi web maupun *mobile*. Pada proses implementasi, Express.js akan mengolah  
30 data lokal sedemikian rupa sehingga dapat dengan mudah diakses apabila diperlukan. Express.js  
31 hanya dapat digunakan untuk membangun aplikasi apabila aplikasi tersebut berjalan didalam  
32 lingkungan Node.js. Oleh karena itu, fitur-fitur yang dimiliki oleh Express.js akan membantu dalam  
33 pembangunan aplikasi berbasis Node.js.

34 Aplikasi yang akan dibangun merupakan aplikasi permainan berbasis web, yang memanfaatkan

1 teknologi *PC* dan *smartphone*. Oleh karena itu, dibutuhkan teknologi yang dapat mengatur  
2 penampilan halaman pada layar *PC* maupun *smartphone*. Teknologi yang digunakan adalah The  
3 Content Template element (`<template>`) [5]. Teknologi ini merupakan bagian dari elemen HTML5,  
4 yang berfungsi untuk menyimpan seluruh elemen-elemen HTML untuk ditampilkan ke layar *browser*  
5 pada *PC* maupun *smartphone*. Didalam satu berkas HTML, elemen `<template>` dapat berjumlah  
6 lebih dari satu. Dengan begitu, beberapa halaman dapat dipilih untuk ditampilkan dalam satu  
7 waktu tertentu, sebelum menampilkan halaman lain yang disimpan oleh `<template>`. Proses  
8 menampilkan halaman ke layar *PC* maupun *smartphone* akan menggunakan JavaScript.

9 Teknologi yang akan membantu dalam penggunaan `<template>` adalah jQuery [6]. Teknologi  
10 ini merupakan pustaka JavaScript yang menyediakan fitur-fitur untuk mengatur berbagai elemen  
11 HTML. Pustaka ini memiliki fitur untuk memanipulasi berkas HTML. Dengan begitu, jQuery dapat  
12 mengatur untuk menampilkan `<template>` mana yang akan ditampilkan ke layar *PC* maupun  
13 *smartphone*.

14 Pada skripsi ini, akan dibuat sebuah aplikasi permainan berbasis web yang memanfaatkan Soc-  
15 ket.io. Selain itu, aplikasi yang dibuat akan memanfaatkan *personal computer (PC)* dan *smartphone*  
16 untuk pengembangan aplikasinya. Para pemain akan mengkoneksikan *smartphone* pada suatu *PC*  
17 yang akan berfungsi sebagai *console*, dan *smartphone* tersebut akan berfungsi sebagai *controller*  
18 untuk memainkan permainannya. Oleh karena itu, Socket.io akan digunakan sebagai koneksi antara  
19 *smartphone* dan *PC* dalam aplikasi permainan yang akan dibangun. Aplikasi permainan akan  
20 dibangun berdasarkan Node.js, sehingga proses eksekusi JavaScript dapat dilakukan pada *server*.  
21 Pengaturan struktur direktori didalam pengembangan aplikasi akan menggunakan Express.js. Did-  
22 lam proses pengaturan elemen grafis yang dibutuhkan didalam aplikasi, teknologi Canvas API akan  
23 digunakan didalam pengembangannya. Elemen `<template>` akan digunakan untuk menampilkan  
24 setiap halaman-halaman web yang dibutuhkan didalam pengembangan aplikasi permainan. Untuk  
25 pengaturan berbagai elemen HTML didalam aplikasi, teknologi jQuery akan digunakan. Aplikasi  
26 permainan akan menggunakan teknologi berbasis web, sehingga untuk memainkannya, *client* harus  
27 memiliki akses internet dan mengakses alamat aplikasi permainan menggunakan *browser*.

## 28 1.2 Rumusan Masalah

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

## 32 1.3 Tujuan

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

## 1 1.4 Batasan Masalah

- 2 Batasan masalah yang dibuat terkait dengan penggerjaan skripsi ini adalah sebagai berikut:
- 3     • Aplikasi permainan yang dibuat merupakan permainan *multiplayer* yang hanya bisa dimainkan  
4         oleh dua orang saja.

## 5 1.5 Metodologi

- 6 Metodologi yang dilakukan dalam penggerjaan skripsi ini adalah sebagai berikut:
- 7     1. Studi literatur mengenai :
- 8         • *Socket.io* sebagai teknologi yang akan menghubungkan *smartphone* dan *PC*.
- 9         • *Canvas API* yang akan digunakan untuk antarmuka permainan.
- 10         • *Node.js* sebagai *web server* dalam pembangunan aplikasi.
- 11         • *Express.js* sebagai *Node.js framework* yang akan digunakan untuk mengatur penyimpanan  
12         data.
- 13         • *jQuery* yang akan digunakan dalam pengaturan elemen HTML.
- 14         • *The Content Template element* yang akan digunakan untuk menampilkan halaman-  
15         halaman HTML.
- 16     2. Menganalisis aplikasi sejenis.
- 17     3. Merancang antarmuka permainan pada *PC* dan *smartphone*. Antarmuka pada *PC* akan  
18         berbeda dengan yang ada di *smartphone*, karena *smartphone* akan bekerja sebagai *controller*  
19         dan *PC* akan bekerja sebagai *console*.
- 20     4. Menyusun cara bermain aplikasi permainan yang dibangun.
- 21     5. Mengimplementasi program aplikasi permainan berbasis web.
- 22     6. Menganalisis *latency* yang dihasilkan pada aplikasi.
- 23     7. Melakukan eksperimen dan pengujian yang melibatkan responden.

## 24 1.6 Sistematika Pembahasan

- 25 Setiap bab dalam skripsi ini memiliki sistematika penulisan yang dijelaskan kedalam poin-poin  
26 sebagai berikut:
- 27     1. Bab 1 : Pendahuluan
- 28         Membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan  
29         masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.

1      2. Bab 2 : Dasar Teori

2      Membahas mengenai teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang  
3      Socket.io, Node.js, Express.js, Canvas API, jQuery, dan The Content Template element.

4      3. Bab 3 : Analisis

5      Membahas mengenai analisa masalah.

6      4. Bab 4 : Perancangan

7      Membahas mengenai perancangan yang dilakukan sebelum melakukan tahapan implementasi.

8      5. Bab 5 : Implementasi dan Pengujian

9      Membahas mengenai implementasi dan pengujian yang telah dilakukan.

10     6. Bab 6 : Kesimpulan dan Saran

11     Membahas hasil kesimpulan dari keseluruhan penelitian ini dan saran-saran yang dapat  
12    diberikan untuk penelitian berikutnya.

1

## BAB 2

2

### LANDASAN TEORI

- 3 Pada bab ini akan dijelaskan landasan teori mengenai Node.js, Express.js, Socket.io, Canvas API,  
4 jQuery, dan The Content Template element.

5 **2.1 Node.js**

- 6 Node.js adalah sebuah *platform* atau lingkungan untuk menjalankan JavaScript pada *web server*  
7 yang dibangun berdasarkan *V8*<sup>1</sup> yang merupakan *engine* JavaScript milik perusahaan *Google* [3].  
8 Node.js memiliki model *event-driven* yang artinya seluruh aksi akan dilakukan berdasarkan suatu  
9 *event* yang dipancarkan, dan *non-blocking* yang artinya suatu aksi dalam Node.js akan langsung  
10 dilakukan tanpa harus menunggu aksi sebelumnya selesai. Teknologi ini menyediakan beberapa  
11 kelas yang berfungsi untuk mengimplementasi fitur-fitur yang dimiliki.  
12 Subbab-subbab berikut akan menjelaskan beberapa kelas yang dimiliki oleh Node.js.

13 **2.1.1 HTTP**

- 14 *HTTP* merupakan suatu *interface* pada Node.js yang digunakan untuk menangani permintaan dari  
15 protokol *HTTP*. *Interface* ini akan menangani protokol *HTTP* dengan tidak melakukan *buffer*, yang  
16 berarti menyimpan sementara seluruh data yang akan dikirimkan pada seluruh permintaan atau  
17 respon.

18 Berikut akan dijelaskan salah satu kelas yang dimiliki oleh *interface* *HTTP*.

19 **1. http.Server**

20 Kelas ini akan menciptakan objek *server* *HTTP*, yang akan menangani setiap permintaan  
21 yang dilakukan oleh *client*. Salah satu *Method* yang dimiliki oleh kelas ini adalah:

22 **• server.listen()**

23 *Method* ini akan memulai *server* *HTTP* untuk melakukan proses *listening* suatu koneksi.  
24 Dengan begitu, *server* akan mengetahui apabila ada *client* yang melakukan permintaan  
25 untuk suatu koneksi.

26 *Interface* *HTTP* memiliki beberapa *method* yang akan membantu dalam proses pengembangan  
27 suatu aplikasi. Salah satu *method* yang dimiliki oleh *HTTP* adalah sebagai berikut:

---

<sup>1</sup><https://v8.dev/>

1     • **http.createServer([options][,requestListener])**

2       **Parameter:**

3           – **options** objek-objek seperti berikut:

4              \* **IncomingMessage**, suatu kelas yang akan menciptakan objek *request* yang merepre-  
5              sentasikan suatu permintaan yang dikirimkan oleh *client*.

6              \* **ServerResponse**, suatu kelas yang akan menciptakan objek *response* yang merepre-  
7              sentasikan suatu respon yang dikirimkan oleh *server*.

8           – **requestListener** fungsi yang akan dieksekusi secara otomatis apabila *event 'request'* di-  
9           pancarkan. *Event* tersebut menandakan bahwa *client* telah melakukan suatu permintaan  
10          kepada *server*

11      **Kembalian:** objek *http.Server*

12     *Method* ini akan membuat objek *http.Server* untuk menangani permintaan dari *client* dan  
13     memberikan respon kepada *client*. Fungsi yang diberikan pada *method* ini akan dipanggil satu  
14     kali pada saat setiap permintaan yang dikirimkan kepada *server*.

15     **2.1.2 Path**

16     *Path* adalah suatu *interface* yang menyediakan fungsi untuk mengatur akses suatu berkas dan  
17     direktori. *Interface* tersebut dapat diakses dengan cara sebagai berikut:

18     const path = require ('path');

Listing 2.1: Akses modul *path*

19     Dengan cara tersebut, maka variabel *path* akan mendapatkan kumpulan fungsi yang dimiliki oleh

20     *Path*. Dengan begitu, variabel tersebut dapat memanggil setiap *method* milik *Path* yang diperlukan.

21     Salah satu method yang dimiliki oleh *Path* adalah sebagai berikut:

22     • **path.join([...paths])**

23       **parameter:**

24           – **...paths**

25              tipe: **String**

26              Urutan suatu lokasi berkas yang akan digunakan.

27      **kembalian:** String

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

30     **2.1.3 Module**

31     Pada aplikasi berbasis Node.js, setiap berkas yang terdapat dalam pembangunan aplikasi dianggap  
32     sebagai modul-modul yang terpisah satu sama lain. Variabel dan fungsi yang terdapat pada satu  
33     berkas, atau modul, hanya dapat digunakan pada satu lingkup modul tersebut. Suatu modul tidak  
34     dapat menggunakan variabel atau fungsi yang terdapat pada modul lainnya. Oleh karena itu,

- 1 apabila variabel dan fungsi yang terdapat pada satu modul dapat digunakan oleh modul yang lain,  
 2 diperlukan cara tertentu. Cara tersebut dapat dilakukan seperti berikut:

3 module.exports = functions || object

Listing 2.2: Proses *export* fungsi dan objek dari satu modul ke modul lain

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

## 8 2.2 Express.js

9 Express.js adalah suatu *framework* untuk membangun aplikasi web yang berjalan berdasarkan  
 10 Node.js [4]. Fitur-fitur yang ada pada Express.js digunakan untuk mengatur struktur direktori dalam  
 11 pengembangan aplikasi web, sehingga untuk proses pemeliharaan aplikasi web dapat dilakukan  
 12 secara efisien. Express.js memiliki beberapa fitur dan proses yang dapat membantu pengembangan  
 13 aplikasi web. Fitur dan proses tersebut akan dijelaskan sebagai berikut:

### 14 • Middleware

15 *Middleware* adalah fungsi yang memiliki akses terhadap seluruh permintaan dari *client*, yang  
 16 dapat memodifikasi permintaan tersebut sebelum mengirimkan respon kembali kepada *client*.  
 17 Fungsi *middleware* bertugas untuk melakukan beberapa hal sebagai berikut:

- 18 – Mengeksekusi perintah kode yang ada didalam fungsi.
- 19 – Memodifikasi objek *request* yang didapat dari *client* dan *response* yang akan diberikan  
 20 kepada *client*.
- 21 – Mengakhiri siklus *request-response*, yang artinya sudah tidak akan ada lagi permintaan  
 22 dari *client* dan respon dari *server*.
- 23 – Memanggil fungsi *next()*, dimana fungsi tersebut akan memanggil fungsi *middleware*  
 24 selanjutnya yang ada didalam program.

25 Contoh bentuk fungsi *middleware* akan dijelaskan sebagai berikut:

```
26 var express = require('express');
27 var app = express();
28
29 app.get('/', function(req, res, next){
30   next();
31 })
```

Listing 2.3: Contoh fungsi *middleware*

32 *Middleware* adalah fungsi yang memiliki parameter *req*, *res*, dan *next*. Bagian-bagian parameter  
 33 akan dijelaskan sebagai berikut:

- **req**, argumen yang merepresentasikan *request* HTTP. Argumen ini dinamakan *req* berdasarkan konvensi.
- **res**, argumen yang merepresentasikan *response* HTTP. Argumen ini dinamakan *res* berdasarkan konvensi.
- **next**, argumen *callback* yang akan dieksekusi berdasarkan kondisi tertentu yang dapat ditentukan. *Callback* adalah fungsi yang berperan sebagai argumen pada parameter fungsi lain. Argumen ini dinamakan *next* berdasarkan konvensi.

### • Routing

*Routing* adalah proses yang menentukan bagaimana suatu aplikasi akan memberikan respon pada saat *client* mengakses *URI* milik aplikasi. Proses ini dapat dilakukan dengan menggunakan *method* milik kelas *Application* 2.2.2 yang sesuai dengan *method* dari protokol HTTP. Berikut merupakan contoh penggunaan proses *routing*:

```

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

app.get('/', function(req, res){
    res.send('GET request to the homepage');
});

app.post('/', function(req, res){
    res.send('POST request to the homepage');
});

```

Listing 2.4: Contoh penggunaan proses *routing*

Proses *routing* dilakukan pada saat ada permintaan dari *client* yang menggunakan *method GET* dan *POST*. Apabila menggunakan *GET*, maka *server* akan memberi respon dengan menampilkan teks '*GET request to the homepage*'. Apabila menggunakan *POST*, maka *server* akan memberi respon dengan menampilkan teks '*POST request to the homepage*'.

Dengan adanya proses *routing*, maka aplikasi dapat menentukan respon yang sesuai dengan permintaan yang dilakukan oleh *client*.

Subbab-subbab berikut akan menjelaskan kelas-kelas yang terdapat pada Express.js

#### 2.2.1 express()

Kelas ini dibutuhkan untuk mengakses seluruh fitur yang disediakan oleh Express.js. Untuk dapat mengakses seluruh fitur Express.js, langkah yang harus dilakukan adalah sebagai berikut:

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

Listing 2.5: Mengakses fitur *express()*

1 Variabel *express* digunakan untuk memuat seluruh fitur yang dimiliki oleh modul '*express*',  
2 sehingga variabel tersebut dapat menggunakan seluruh fitur yang tersedia. Dengan melakukan lang-  
3 kah tersebut, fitur-fitur yang terdapat pada Express.js sudah dapat digunakan untuk pengembangan  
4 aplikasi web.

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

6 • **express.static(root, [options])**

7     **Parameter:**

8         – **root**

9             tipe: **String**

10          Menentukan direktori sumber utama yang akan digunakan untuk menyediakan berkas  
11          yang memiliki sifat statis. Berkas yang memiliki sifat statis dapat dikirimkan kepada  
12          *client* tanpa harus dibangkitkan, dimodifikasi, atau dilakukan proses tertentu terlebih  
13          dahulu.

14         – **options**

15             Suatu objek seperti berikut:

- 16                 \* **dotfiles**, menentukan bagaimana mengatasi suatu *dotfiles* (suatu berkas atau direk-  
17                 tori yang dimulai dengan tanda ".").

18      *Method* ini menyediakan cara agar dapat mengakses berkas yang memiliki sifat statis. Berkas  
19      tersebut disimpan dalam suatu direktori lokal yang dapat diakses didalam proses pengem-  
20      bangan aplikasi web.

21 • **express.Router([options])**

22     **Parameter:**

23         – **options** merupakan parameter pilihan yang akan menentukan perilaku dari *server* pada  
24         saat diakses oleh *client*. Parameter dapat berupa objek sebagai berikut:

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

26      *Method* ini akan membuat objek *router* [2.2.4](#) yang dapat digunakan untuk menangani permintaan  
27      dari *client* pada saat *client* mengakses *URI* tertentu.

28 **2.2.2 Application**

29 Kelas ini akan mengatur bagaimana perilaku aplikasi terhadap berbagai permintaan yang dilakukan  
30 oleh *client*. Objek yang diciptakan dari kelas *Application* dapat melakukan beberapa hal seperti  
31 berikut:

- 32     • Proses *routing* terhadap permintaan HTTP.  
33     • Mengatur berjalannya *middleware*.  
34     • Melakukan proses *rendering* pada berkas HTML.  
35     • Mendaftarkan *template engine* tertentu, yang digunakan untuk mengubah elemen-elemen  
36         HTML.

1        Objek dari kelas *Application* dapat diciptakan dengan langkah seperti berikut:

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

Listing 2.6: Mengakses fitur *application*

4        Baris pertama dari potongan kode tersebut berarti variabel *express* yang memanggil modul  
 5        'express' agar dapat mengakses fungsi-fungsi yang ada pada modul tersebut. Sedangkan baris kedua,  
 6        Objek *app* memanggil fungsi *express()* yang telah didapatkan dari variabel *express*. Dengan begitu,  
 7        variabel *app* dapat mengakses fungsi-fungsi yang ada didalam pengembangan aplikasi web.

8        Kelas ini memiliki beberapa *method* sebagai berikut:

- 9        • **app.set(name, value)**

10      **Parameter:**

- 11        – **name**, nama yang akan digunakan untuk menentukan perilaku dari suatu *server*.
- 12        – **value**, nilai yang akan ditetapkan pada parameter *name*.

13      **Kembalian:** -

14      Method ini akan menetapkan suatu nilai yang disimpan pada variabel *value*, kepada suatu  
 15        nama yang disimpan pada variabel *name*. Beberapa contoh nama yang dapat digunakan  
 16        adalah *views*, dan *view engine*. Nama-nama tersebut akan menentukan jenis *view engine* apa  
 17        yang akan dipakai didalam proses pengembangan aplikasi.

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

19      **Parameter:**

- 20        – **path**, suatu *String* yang merepresentasikan *path* yang akan ditangani oleh *middleware*.
- 21        – **callback**, fungsi yang disimpan didalam parameter fungsi lain, yang dapat menjalankan  
        perintah-perintah tertentu.

23      **Kembalian:** -

24      Method ini akan menghubungkan *middleware* dengan *path* yang sudah ditentukan. Dalam  
 25        implementasi *method* ini, urutan penempatan pada baris kode sangat berpengaruh. Setelah  
 26        *app.use()* dieksekusi, maka suatu permintaan tidak akan mengeksekusi *middleware* yang ada  
 27        dibawah baris kode *app.use()*.

### 28      2.2.3 Response

29      Sebuah objek dari kelas *Response* akan merepresentasikan respon HTTP yang dikirim oleh Express  
 30        pada saat menerima permintaan HTTP. Didalam pengembangan aplikasi web, objek dari kelas  
 31        *Response* cenderung direpresentasikan dengan variabel *res*.

32      Salah satu *method* yang terdapat pada kelas *Response* adalah sebagai berikut :

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

34      **Parameter:**

- 1     – **view**, suatu *String* yang menunjukan *path* dari berkas *view*.
- 2     – **locals**, suatu objek yang memiliki properti yang menunjukan variabel lokal yang ada pada berkas *view*.
- 3     – **callback**, fungsi yang disimpan didalam parameter fungsi lain, yang dapat menjalankan perintah-perintah tertentu.

6     *Method* ini berfungsi untuk melakukan proses *render* pada berkas *view* dan mengirim hasil  
7     perubahan berkas tersebut kepada *client*.

#### 8     2.2.4 Router

9     *Router* adalah kelas yang memiliki suatu objek yang dapat mengatur bagaimana *URI (Uniform*  
10    *Resource Identifier*), yang merupakan alamat suatu aplikasi melakukan respon pada saat *client*  
11    mengakses *URI* tersebut. Proses tersebut dapat disebut sebagai *routing*.

12    Proses *routing* akan menentukan fungsi *callback* yang akan menangani permintaan dari *client*.  
13    Fungsi tersebut akan dipanggil saat ada permintaan dari *client* atau ada pemanggilan dari suatu  
14    *method* milik HTTP. Dengan kata lain, suatu aplikasi akan selalu melakukan proses *listen*, yang  
15    berarti menunggu apabila ada permintaan dari *client* yang kemudian dicocokan dengan *route* yang  
16    sesuai. Apabila permintaan *client* dengan *route* yang ada telah cocok, maka fungsi *callback* yang  
17    telah ditetapkan akan dieksekusi.

18    Salah satu *method* yang dimiliki oleh kelas ini adalah sebagai berikut:

##### 19    • **router.METHOD()**

20    *Method* ini akan menyediakan fungsionalitas proses *routing* pada aplikasi Express. *METHOD*  
21    merupakan fungsi-fungsi yang merepresentasikan permintaan HTTP, seperti *GET* dan *POST*.  
22    Contoh penggunaan dari *method* tersebut adalah sebagai berikut:

```
23               var express = require('express');
24               var router = express.Router();
25
26               router.get('/', function(req, res, next){
27                   res.send('GET routing to the homepage');
28       });
29
30               router.post('/', function(req, res, next){
31                   res.send('POST routing to the homepage');
32       });
```

33    Untuk dapat menggunakan fitur yang disediakan, maka variabel *router* akan melakukan proses  
34    instansiasi dengan menggunakan *express.Router()*. *Method* *router.get()* akan menampilkan pesan  
35    *GET routing to the homepage* apabila *client* melakukan permintaan *GET*. Sedangkan *method*  
36    *router.post()* akan menampilkan pesan *POST routing to the homepage* kepada *client* apabila *client*  
37    melakukan permintaan *POST*.

## 1 2.3 Socket.io

2 Socket.io adalah teknologi yang memungkinkan sebuah aplikasi untuk melakukan komunikasi dua  
 3 arah secara *real-time* [1]. Socket.io dapat berjalan di setiap *platform*, *browser*, dan gawai.

4 Teknologi Socket.io memanfaatkan teknologi *socket* dalam proses implementasinya. *Socket*  
 5 merupakan suatu *endpoint* didalam hubungan komunikasi dua arah antara dua program yang  
 6 berjalan didalam jaringan tertentu. *Endpoint* adalah suatu entitas yang merupakan titik akhir dari  
 7 suatu komunikasi, dimana entitas tersebut terdiri dari kombinasi alamat *IP* (*Internet Protocol*) dan  
 8 nomor *port*. *Socket* akan terikat dengan nomor *port* agar *TCP* (*Transmission Control Protocol*), yang  
 9 mengatur koneksi internet, dapat mengidentifikasi suatu aplikasi untuk data yang akan dikirimkan.

10 Sebelum dapat menggunakan Socket.io, Node.js harus sudah tersedia pada sistem komputer.  
 11 Apabila hal tersebut sudah dilakukan, maka proses *install* Socket.io dapat dilakukan dengan  
 12 menggunakan *command line tools* atau sejenisnya. Langkah tersebut adalah sebagai berikut:

13 `npm install socket.io`

Listing 2.7: install Socket.io

14 Dengan melakukan proses tersebut, aplikasi yang akan dibuat sudah dapat mengakses fitur-fitur  
 15 yang dimiliki oleh Socket.io.

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

### 18 2.3.1 Server API

19 Kelas-kelas yang ada pada *Server API* digunakan untuk menangani proses yang terjadi didalam  
 20 *server*. Berikut akan dijelaskan kelas-kelas yang dimiliki oleh *Server API*.

#### 21 1. Server

22 Kelas ini merupakan kelas inti untuk dapat menangani proses yang terjadi dalam *server*  
 23 Socket.io. Kelas ini memiliki konstruktor seperti berikut:

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

25 **Parameter:**

- 26 – **httpServer**

27 tipe: **http.Server**

28 Objek *server* HTTP yang akan diintegrasikan.

- 29 – **options**

30 tipe: **Object**

31 Parameter ini dapat berupa berbagai jenis objek. Salah satu dari objek tersebut  
 32 adalah sebagai berikut:

- 33 \* **path**

34 tipe: **String**

35 Nama dari *path* yang akan ditangkap oleh *server* (contoh: */socket.io*).

36 Salah satu *method* yang dimiliki oleh kelas ini adalah sebagai berikut:

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

2       **Parameter:**

3           – **port**

4              tipe: **Integer**

5              Nomor yang akan digunakan untuk melakukan koneksi kepada *server*.

6           – **options**

7              tipe: **Object**

8              Parameter ini dapat berupa berbagai jenis objek. Salah satu objek yang dapat  
9              dimasukan adalah sebagai berikut:

10             \* **serveClient**, parameter ini memiliki tipe *boolean*, dengan nilai *default* yaitu  
11             *true*.

12     Method ini akan melakukan koneksi kepada *server* dengan menuju nilai *port* yang terdapat  
13     pada parameter.

## 14    2. Namespace

15    Kelas ini merepresentasikan kumpulan *socket* yang terkoneksi satu sama lain didalam lingkup  
16    yang diidentifikasi oleh nama *path*. *Namespace* merupakan suatu *endpoint* atau *path* yang  
17    ditetapkan kepada suatu *socket* tertentu. *Client* akan selalu terhubung ke /, yang merupakan  
18    *namespace* utama, kemudian dapat terhubung ke *namespace* lain saat berada dalam koneksi  
19    yang sama (contoh: /chat).

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

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

22     Berfungsi untuk memancarkan suatu *event* pada seluruh *client* yang terhubung.

23       **Parameter:**

24           – **eventName**

25              tipe: **String**

26              Nama dari suatu *event* yang akan dipancarkan.

27           – **args**

28              Argumen tertentu yang akan dikirimkan bersamaan dengan *event* yang dipancarkan.

- 29     ● **namespace.to(room)**

30     Berfungsi untuk memancarkan *event* hanya kepada *client* yang berada didalam *room*  
31     tertentu.

32       **Parameter:**

33           – **room**

34              tipe: **String**

35              kode suatu *room* milik beberapa *client* yang terkoneksi ke Socket.io.

## 36    3. Socket

37    Kelas ini merupakan kelas yang mendasar untuk berinteraksi dengan *client*. Suatu objek  
38    *socket* akan selalu terhubung kepada *namespace* tertentu.

39    Beberapa properti yang dimiliki oleh kelas ini adalah sebagai berikut:

1     • **socket.id**

2       Tanda pengenal unik milik *client* yang terkoneksi ke Socket.io.

3     • **socket.rooms**

4       Suatu *String* yang menjadi identifikasi dimana *client* berada didalamnya.

5       Salah satu *method* yang dimiliki oleh kelas ini adalah sebagai berikut:

6     • **socket.join(room[, callback])**

7       Berfungsi untuk menambah *client* kedalam *room*.

8       **Parameter:**

9           – **room**

10          tipe: **String**

11          Nama *room* yang dapat memasukan beberapa *client*.

12           – **callback**

13          tipe: **Function**

14          Fungsi yang menangani proses tertentu yang menjadi masukan didalam parameter fungsi lain.

16     **2.3.2 Client API**

17     *Client API* digunakan untuk menangani proses pengaturan koneksi yang terjadi pada bagian *client*.

18     Agar dapat menggunakan fitur-fitur yang tersedia, *client* harus menambahkan *URL* pada JavaScript didalam HTML yang bersangkutan. Hal tersebut dapat dilakukan seperti berikut:

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

Listing 2.8: Mengakses fitur Socket.io untuk *client*

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

22     1. **IO**

23     Kelas ini merupakan kelas inti yang akan digunakan untuk mendapatkan fitur-fitur yang akan digunakan pada bagian *client*. Untuk dapat menggunakan fungsi yang ada pada kelas *IO*, dapat dilakukan langkah seperti berikut:

Listing 2.9: Mendapatkan fitur-fitur Socket.io pada bagian *client*

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

27     Dengan melakukan langkah tersebut, maka bagian *client* akan dapat mengakses seluruh fitur yang diperlukan untuk membangun koneksi ke Socket.io.

29     *Method* yang dimiliki oleh kelas ini adalah sebagai berikut:

30     • **io([url][, options])**

31       **Parameter:**

32           – **url**

33          tipe: **String**

34          Nama *URL*.

1       – **options**

2           tipe: **Object**

3           Parameter ini dapat berupa beberapa jenis objek. Salah satu jenis objek tersebut  
4           adalah sebagai berikut:

- 5           \* **path**, nama suatu *path* yang akan dituju.

6       **Kembalian:** Objek *Socket*

7       *Method* ini berfungsi untuk melakukan proses inisialisasi objek *Socket* yang digunakan  
8           untuk melakukan koneksi ke Socket.io.

9       2. **Socket**

10      Kelas ini digunakan untuk membuat objek *Socket* yang akan digunakan untuk melakukan  
11           koneksi ke Socket.io pada bagian *client*.

12      Salah satu properti yang dimiliki oleh kelas ini adalah:

13       ● **socket.id**

14      Identifikasi unik yang dimiliki oleh satu *client* untuk satu sesi. Properti ini akan  
15           mempunyai nilai segera setelah koneksi terhubung, dan akan diperbarui setelah melakukan  
16           koneksi ulang.

17      Beberapa *method* yang dimiliki oleh kelas ini adalah:

18       ● **socket.on(eventName, callback)**

19       **Parameter:**

- 20           – **eventName**, nama *event* yang dipancarkan.

- 21           – **callback**, fungsi yang akan menangani *event* yang dipancarkan.

22      *Method* ini akan menangkap *event* yang dipancarkan oleh *server*, yang kemudian akan  
23           mengeksekusi fungsi *callback* untuk melakukan perintah tertentu.

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

25       **Parameter:**

- 26           – **eventName**

27           tipe: **String**

28           Nama *event*.

- 29           – **args**

30           Argumen tambahan (opsional).

- 31           – **ack**

32           tipe: **Function**

33           Fungsi tambahan (opsional).

34      *Method* ini akan memancarkan suatu *event* dengan nama *eventName* kepada *server*,  
35           dengan argumen yang telah ditentukan sebelumnya.

36      Beberapa *event* yang ada pada kelas ini adalah sebagai berikut:

1     • **connect**

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

4     • **disconnect**

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

6 **2.4 Canvas API**

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

- 11   1. Menambahkan *tag <canvas>* pada berkas HTML, dan menambahkan *id* yang akan digunakan  
12   pada berkas JavaScript.

13   <canvas id="canvas"></canvas>

Listing 2.10: *tag Canvas API* pada JavaScript

- 14   2. Membuat variabel untuk mendapatkan konteks *rendering* yang berguna untuk memodifikasi  
15   elemen didalam *canvas* dan fungsi-fungsi menggambar agar dapat menampilkan sesuatu pada  
16   <canvas>.

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

20
21   // Variabel yang akan mendapatkan fungsi-fungsi menggambar
22   var ctx = canvas.getContext('2d');
```

Listing 2.11: Membuat objek *canvas*

23   Dengan melakukan langkah-langkah tersebut, maka fungsi-fungsi yang dimiliki Canvas API  
24   dapat digunakan.

25   Subbab-subbab berikut akan menjelaskan tentang beberapa elemen yang tersedia didalam  
26   Canvas API.

27 **2.4.1 Animation**

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

31 **Langkah Dasar Animasi**

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

1    **1. Menghapus isi *canvas***

2    Untuk memulai animasi, langkah pertama yang harus dilakukan adalah menghapus seluruh  
3    elemen yang berada di dalam *canvas*. Langkah tersebut dilakukan untuk menghilangkan  
4    bentuk atau elemen lain yang sudah digambar sebelumnya didalam *canvas*, sehingga dapat  
5    menggambar suatu *frame*, yang merupakan tempat dimana animasi akan berlangsung.

6    **2. Menyimpan *state* milik *canvas***

7    *State* adalah kondisi saat ini dari objek *canvas*. Apabila akan dilakukan perubahan seperti  
8    merubah warna bentuk, transformasi bentuk, atau hal lainnya yang akan menyebabkan  
9    perubahan pada *state* dari *canvas*, maka *state* harus disimpan setiap ada perubahan.

10   **3. Menggambar elemen untuk animasi**

11   Pada langkah ini, elemen yang akan diproses untuk animasi akan digambar kedalam *canvas*.  
12   Elemen yang digambar dapat berupa bentuk bangunan, gambar, maupun tulisan.

13   **4. Mengembalikan *state* dari *canvas***

14   Beberapa *state* dari *canvas* yang sudah disimpan, harus dikembalikan seluruhnya agar kembali  
15   ke *state* pertama sebelum ada perubahan apapun. Hal tersebut dilakukan sebelum menggambar  
16   *frame* baru untuk proses animasi selanjutnya. Proses animasi akan diulang kembali mulai  
17   dari langkah pertama.

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

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

22   **Parameter:**

23   – **func**

24   Suatu fungsi yang akan dieksekusi setiap penundaan milidetik.

25   – **delay**

26   Waktu dalam milidetik.

27   – **param1,...,paramN**

28   Parameter tambahan untuk fungsi apabila waktu telah habis.

29   **Kembalian:**

30   **intervalID** identifikasi unik milik suatu interval.

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

- 34   • **clearInterval(intervalID)**

35   **Parameter:**

36   – **intervalID** identifikasi milik suatu interval yang akan diberhentikan.

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

3     • **requestAnimationFrame(callback)**

4       **Parameter:**

5           – **callback**

6       Fungsi yang akan dipanggil saat harus memperbarui animasi dan menggambar *frame*  
7       selanjutnya.

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

11    **2.4.2 canvasRenderingContext2D**

12    *canvasRenderingContext2D* adalah interface yang digunakan untuk menggambar persegi panjang,  
13    teks, gambar, dan objek-objek lain kedalam elemen *canvas*. *CanvasRenderingContext2D* menyediakan konteks *2D rendering* untuk suatu elemen *<canvas>*, yang berguna untuk memodifikasi  
14    objek dua dimensi didalam *canvas*. Untuk mendapatkan objek dari *interface* ini, harus memanggil  
15    *getContext()* didalam elemen *<canvas>*, dengan memberi '2d' sebagai argumen. Berikut merupakan  
16    contoh penggunaannya:

Listing 2.12: mendapatkan konteks *canvas*

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

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

20     • **CanvasRenderingContext2D.globalCompositeOperation**

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

24     Berikut contoh tipe yang dapat digunakan:

25           – **destination-over**, tipe ini akan membuat suatu elemen yang baru akan digambar pada  
26           posisi dibelakang konten *canvas* yang telah ada sebelumnya.

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

28     • **CanvasRenderingContext2D.clearRect(x, y, width, height) Parameter:**

29           – **x**

30       Koordinat x yang menandakan titik awal persegi.

31           – **y**

32       Koordinat y yang menandakan titik awal persegi.

- 1     – **width**
- 2       Lebar persegi.
- 3     – **height**
- 4       tinggi persegi.

5     *Method* ini akan menghapus seluruh elemen yang telah digambar sebelumnya pada *canvas*  
6     dengan membentuk suatu persegi. *Method* ini akan menggambar koordinat titik awal (*x*, *y*)  
7     dengan lebar dan tinggi yang sudah ditentukan oleh *width* dan *height*.

8     • **CanvasRenderingContext2D.drawImage(image, dx, dy)**

9       **Parameter:**

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

13     *Method* ini akan menyediakan cara untuk menggambar suatu elemen *image* atau gambar pada  
14     *canvas*.

15     • **CanvasRenderingContext2D.save()**

16     *Method* ini akan menyimpan seluruh *state* dari *canvas* dengan menaruh *state* tersebut kedalam  
17     suatu *stack* yang sudah diatur didalam elemen *<canvas>*. Proses ini dilakukan apabila  
18     telah terjadi perubahan bentuk atau komposisi yang menyebabkan *state* dari *canvas* berubah,  
19     sehingga *state* sebelumnya tetap akan tersimpan.

20     • **CanvasRenderingContext2D.restore()**

21     *Method* ini akan mengembalikan *state* yang baru saja disimpan saat menggunakan *method*  
22     *save()* dengan mengeluarkannya dari tumpukan paling atas suatu *stack*. Apabila *stack* tersebut  
23     kosong, maka *method* ini tidak melakukan apapun.

24     

## 2.5 jQuery

25     jQuery adalah pustaka JavaScript yang menyediakan fitur-fitur untuk mengatur berbagai elemen  
26     HTML [6]. Pustaka ini memiliki fitur-fitur seperti memanipulasi berkas HTML, menangani suatu  
27     *event*, dan mengatur jalannya animasi. jQuery dapat menyediakan fitur-fitur untuk menangani  
28     berbagai hal tersebut yang berjalan di berbagai *browser* yang berbeda.

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

30     

### 2.5.1 .submit(handler)

31       **Parameter:**

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

33     *Method* ini akan menyambungkan fungsi yang mengatasi suatu *event* yang memiliki nama "submit"  
34     yang merupakan *event* dari JavaScript, atau memancarkan *event* tersebut kepada elemen tertentu.

1 **2.5.2 .val()**

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

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

4

5 **2.5.3 .html()**

6 **Kembalian:** *String*

7 *Method* ini akan mendapatkan konten HTML dari elemen pertama yang ada didalam kumpulan  
8 elemen-elemen yang sesuai.

9 **2.5.4 .preventDefault()**

10 Apabila *method* ini dieksekusi, maka aksi *default* dari suatu *event* tidak akan dieksekusi. Aksi  
11 *default* yang dilakukan adalah melakukan *reload* pada halaman web.

12 **2.6 The Content Template element**

13 HTML Content Template element (<template>) adalah suatu mekanisme untuk menyimpan konten  
14 milik *client* agar konten tersebut tidak ditampilkan pada saat memuat halaman [5]. Konten tersebut  
15 dapat dipakai pada saat web dijalankan dengan menggunakan JavaScript.

16 Berikut merupakan contoh penggunaan dari elemen <template>:

Listing 2.13: Contoh penggunaan elemen <template>

```
17 <html>
18 <head>
19 <title>Home</title>
20 </head>
21 <body>
22 <p>Hello World!</p>
23
24 <template id="stage">
25
26 </template>
27
28 <template id="home_page">
29 <p>Hello from template element!</p>
30 </template>
31
32 </body>
33 </html>
```

34 Apabila halaman HTML ini dimuat, maka yang akan ditunjukan dihalaman adalah teks "Hello  
35 World!". Seluruh elemen yang ada didalam elemen <template> tidak akan ditampilkan dihalaman

- 1 web. Agar seluruh isi dari elemen <template> ditampilkan, maka diperlukan JavaScript. Berikut
- 2 merupakan contoh kode JavaScript untuk memuat elemen <template>.

Listing 2.14: Memuat elemen <template>

```
3 <script>
4 // variabel yang menyimpan elemen dari <template> stage
5 var bg = $("#stage");
6
7 // variable ini menyimpan seluruh isi dari template home_page
8 var home = $("#home_page").html();
9
10 // variable bg akan memuat template dari home_page
11 bg.html(home);
12
13 </script>
14 Agar dapat menampilkan konten yang berada didalam <template>, maka method .html() milik
15 jQuery akan digunakan. Method tersebut digunakan pada bagian bg.html(home), dimana method
16 tersebut akan memuat seluruh isi variabel home.
```



1

## BAB 3

2

### ANALISIS

- 3 Pada bab ini akan dijelaskan mengenai analisis aplikasi sejenis, analisis alur permainan Finger For  
4 Life, analisis pengembangan web, analisis *use case*, analisis arsitektur Finger For Life, dan analisis  
5 Socket.io.

#### 6 3.1 Analisis Aplikasi Sejenis

- 7 Salah satu aplikasi sejenis permainan berbasis web dengan memanfaatkan *smartphone* sebagai  
8 pengendali adalah AirConsole. Aplikasi tersebut memanfaatkan *browser*, *smartphone*, *PC*, dan juga  
9 jaringan internet untuk dapat menggunakannya. Aplikasi ini dikembangkan oleh N-Dream AG<sup>1</sup>.

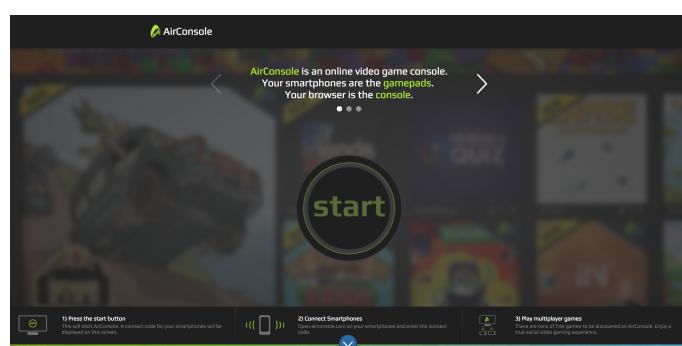
##### 10      **Analisis AirConsole**

11 AirConsole merupakan permainan berbasis web dimana *browser* pada *smartphone* dapat melalui  
12 koneksi ke *browser* pada *PC*. Pada aplikasi ini, terdapat berbagai macam permainan yang  
13 dapat dipilih oleh pemain. Untuk dapat memainkan aplikasi tersebut, pemain harus membuka  
14 alamat web <https://www.airconsole.com/> pada browser di *PC* dan juga di *smartphone*.

15 Analisis dilakukan dengan cara berikut:

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

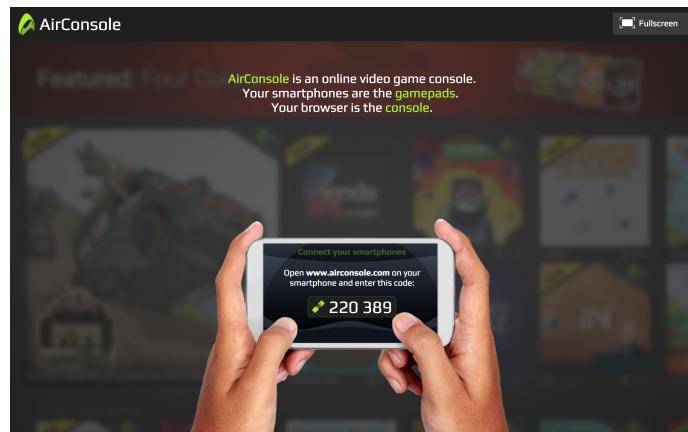
19 Pada halaman awal web di *PC*, pemain diminta untuk menekan tombol *start* yang ada pada gambar  
20 berikut:



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

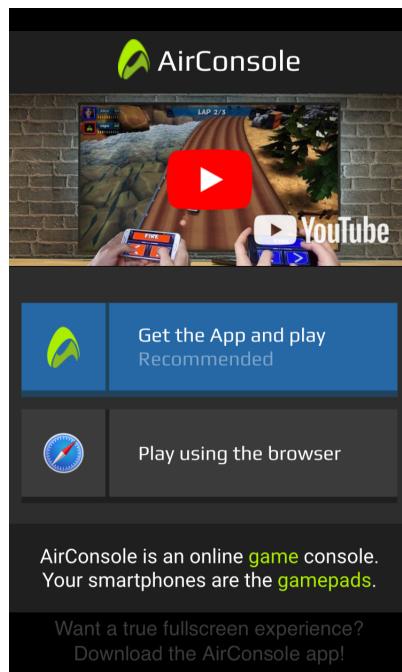
<sup>1</sup> <https://www.airconsole.com/>, diakses 4 Desember 2017

- 1 Setelah tombol *start* ditekan, maka akan muncul halaman berikutnya yang menunjukan kode  
 2 yang harus dimasukan oleh pemain pada *browser* di *smartphone*.



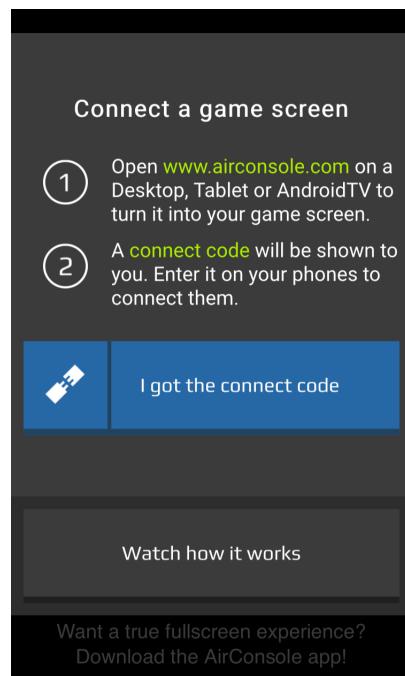
Gambar 3.2: Kode yang harus dimasukan oleh pemain pada *smartphone*.

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



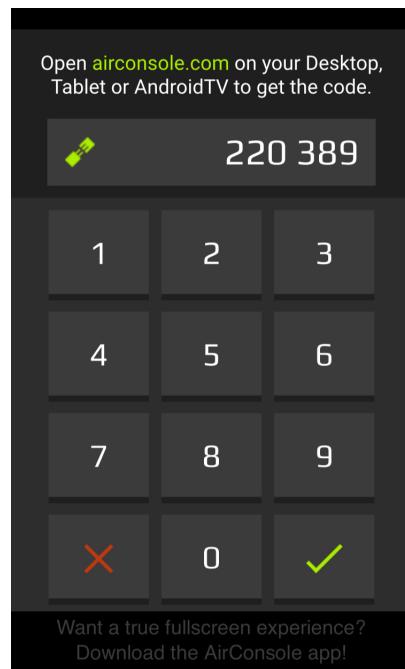
Gambar 3.3: Halaman awal Airconsole pada *smartphone*.

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



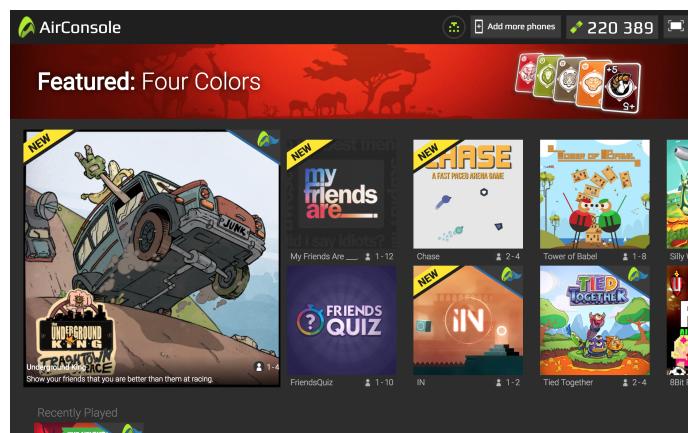
Gambar 3.4: Pemain diminta untuk memasukan kode yang sudah didapatkan pada *PC*.

- <sup>1</sup> Setelah menekan tombol tersebut, pemain dapat mulai memasukan kode yang sudah didapatkan.
- <sup>2</sup> Kode ini bertujuan untuk proses verifikasi, sehingga para pemain yang dapat bermain dalam satu
- <sup>3</sup> sesi yang sama, hanya para pemain yang mengetahui kode tersebut.

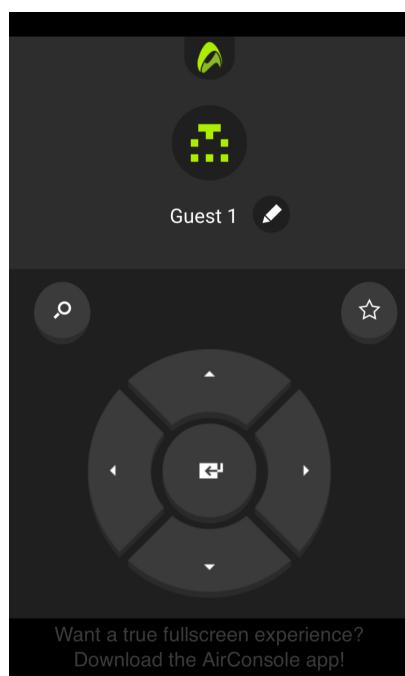


Gambar 3.5: Memasukan kode yang sudah didapatkan pada *PC*.

- <sup>4</sup> Setelah pemain memasukan kode, maka halaman web di *PC* dan *smartphone* akan berubah.
- <sup>5</sup> Pada *PC*, halaman akan menunjukan berbagai jenis permainan yang dapat dipilih. Pada *smartphone*,
- <sup>6</sup> halaman akan berubah menjadi pengendali permainan, dimana pemain dapat menggerakan halaman
- <sup>7</sup> yang ada di *PC* dengan menggunakan *smartphone*.

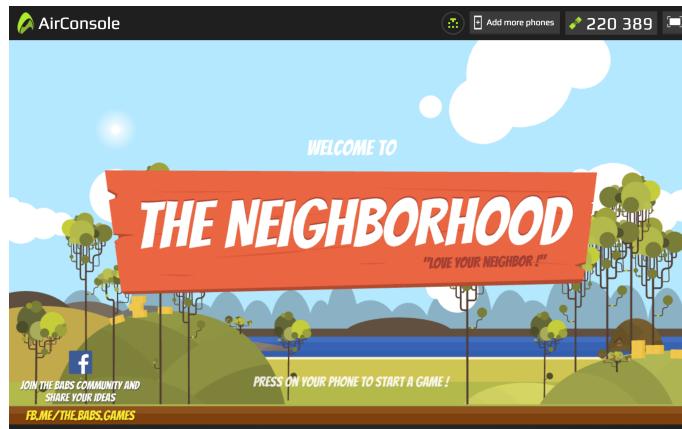


Gambar 3.6: Halaman pada *PC* yang menunjukan berbagai permainan yang dapat dipilih.



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

<sup>1</sup> 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*.

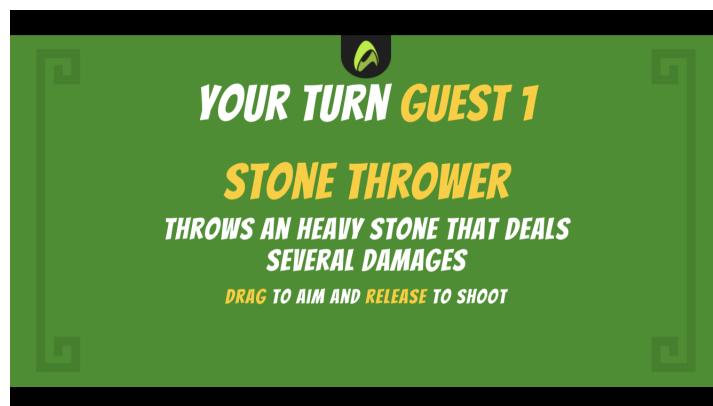


Gambar 3.9: Halaman awal permainan The Neighborhood pada *smartphone*.

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

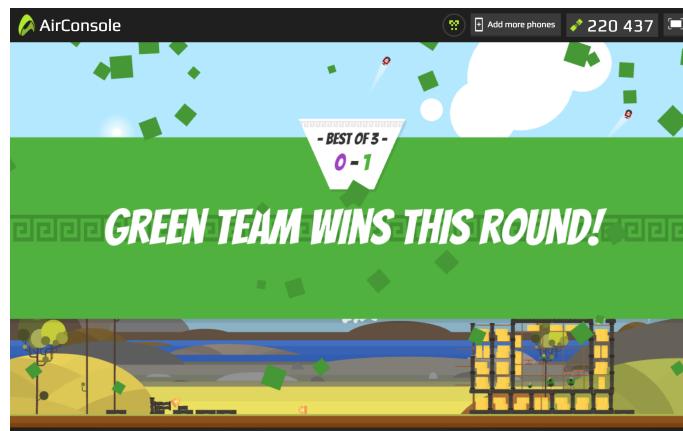


Gambar 3.10: Halaman pada *PC* dimana permainan sedang berlangsung.

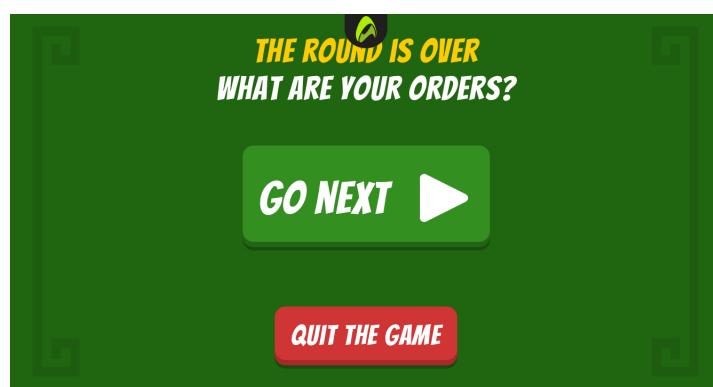


Gambar 3.11: Halaman pada *smartphone* dimana permainan sedang berlangsung.

- 1 Pemain dapat memilih untuk keluar dari permainan atau melanjutkan permainannya kembali
- 2 setelah menyelesaikan satu level permainan.



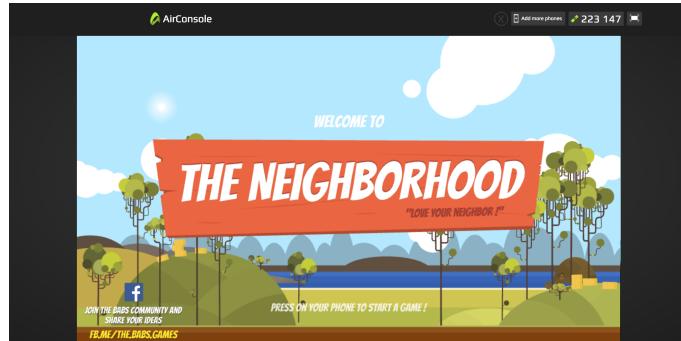
Gambar 3.12: Halaman pada *PC* apabila permainan sudah dimenangkan.



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

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

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



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

## 7 3.2 Analisis Alur Permainan Finger For Life

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

- 13 • Satu Komputer *PC*.  
14 • Dua *smartphone*.  
15 • Jaringan Internet.

16 Para pemain harus menyediakan satu perangkat komputer *PC*, dimana *PC* tersebut akan  
17 berfungsi sebagai *console*, dan dua *smartphone* yang akan berfungsi sebagai *controller*. *PC* akan  
18 memiliki peran sebagai *host* yang menciptakan suatu *room*, dimana *room* tersebut merupakan sebuah  
19 kode ruang permainan dimana kedua pemain dapat bergabung kedalamnya. Ketiga perangkat  
20 tersebut akan membutuhkan jaringan internet untuk dapat mengakses alamat web yang harus  
21 dituju. Setelah ketiga perangkat tersebut tersedia, maka para pemain dapat membuka *browser*  
22 untuk mengakses alamat web permainan Finger For Life. Alamat permainan tersebut adalah  
23 <http://fingerforlife.herokuapp.com/>. Agar para pemain dapat mulai memainkan permainan  
24 ini, ada beberapa tahap yang harus dilakukan. Tahap-tahap yang dilakukan akan dibagi menjadi  
25 dua, yaitu yang terjadi pada bagian *PC*, dan pada *smartphone*. Tahap-tahap tersebut akan dijelaskan  
26 sebagai berikut:

27 1. Permintaan Bergabung

**1 PC**

2 Saat *client* mengakses alamat web Finger For Life pada *PC*, *client* akan ditujukan kehalaman  
3 utama. Halaman ini terdapat tombol *start* yang harus ditekan oleh *client* agar dapat bergabung  
4 kedalam permainan. Setelah tombol *start* ditekan, *client* akan ditujukan kehalaman permintaan bergabung.  
5

6 Pada tahap ini, halaman pada *PC* akan menunjukan perintah untuk mengakses alamat web  
7 Finger For Life pada *browser* di kedua *smartphone*. Halaman ini pun akan menunjukan  
8 kode yang harus dikirimkan oleh para pemain. Kode tersebut merupakan *room* yang akan  
9 menyimpan tiga *client* pada saat permainan dimulai. Ketiga *client* tersebut adalah *host*,  
10 pemain pertama, dan pemain kedua. Para pemain harus mengirimkan kode tersebut kepada  
11 *host*, sehingga *smartphone* pemain dapat tersambung dengan *PC*. Kode yang tersedia berfungsi  
12 untuk proses verifikasi, apakah kode yang dikirimkan dari *smartphone* sama dengan yang ada  
13 di *PC* atau tidak. Proses tersebut menentukan apakah *client* dapat bergabung kedalam *room*  
14 atau tidak.

**16 Smartphone**

17 Saat *client* mengakses alamat web Finger For Life pada *smarthonne*, *client* akan ditujukan  
18 kehalaman utama. Halaman ini terdapat tombol *join* yang harus ditekan oleh *client* agar  
19 dapat bergabung kedalam permainan. Setelah tombol *join* ditekan, *client* akan ditujukan  
20 kehalaman permintaan bergabung.

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

**27 2. Memilih Karakter****29 PC**

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

35 **Smartphone** Halaman ini akan menunjukan daftar karakter yang dapat dipilih oleh para  
36 pemain. Karakter yang dipilih akan ditunjukan dilayar *PC*. Apabila pemain akan menetapkan  
37 karakter yang dipilih, maka pemain dapat menekan tombol *choose*. Setelah kedua pemain  
38 menetapkan karakter masing-masing, maka akan dituju kehalaman selanjutnya.

**39 3. Memainkan Permainan****40 PC**

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

6 **Smartphone**

7 Pada saat *countdown* dilakukan halaman *PC*, halaman *smartphone* akan menampilkan instruksi  
8 untuk memainkan permainan ini. Instruksi tersebut menunjukan untuk menekan tombol kaki  
9 kiri dan kanan untuk menggerakan karakter yang ada di layar *PC*. Setelah *countdown* selesai,  
10 maka halaman akan menampilkan dua telapak kaki yang berfungsi sebagai tombol. Untuk  
11 dapat menggerakan karakter, pemain harus menekan kedua tombol tersebut secara berulang.  
12 Apabila sudah ada pemain yang mencapai garis akhir, maka halaman akan dituju ke halaman  
13 selanjutnya.

14 **4. Mengakhiri Permainan**

15 **PC**

16 Halaman ini akan menampilkan pemenang yang berhasil mencapai garis akhir lebih dulu.  
17 Halaman ini menampilkan podium yang menempatkan para pemenang sesuai posisi nomor  
18 urut masing-masing. Pada tahap ini, karakter milik pemenang akan ditampilkan di podium  
19 nomor satu, dan pemain selanjutnya dinomor dua. Untuk mengakhiri permainan, pemain  
20 harus menekan tombol *exit* yang terdapat di halaman ini. Apabila tombol tersebut ditekan,  
21 maka halaman akan menuju kembali ke halaman utama dan koneksi Socket.io pun akan  
22 terputus.

23 **Smartphone**

24 Pada halaman ini akan ditampilkan teks yang menunjukan posisi pemenang. Apabila pemain  
25 lebih dulu mencapai garis akhir, maka akan ditampilkan teks *YOU WIN*. Pemain yang kalah  
26 akan ditampilkan teks *YOU LOSE*. Setelah tombol *exit* dilayari *PC* ditekan, maka permainan  
27 telah berakhir, dan halaman akan menuju kembali ke halaman utama.

29 **3.3 Analisis Pengembangan Web**

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

33 **1. Node.js**

34  
35 (a) **HTTP Interface** HTTP digunakan pada bagian *server*. Web yang akan dikembangkan  
36 akan menggunakan *server* HTTP untuk menyediakan akses ke alamat web yang dapat  
37 diakses oleh *client*. HTTP *server* akan diintegrasikan dengan Socket.io, dimana HTTP  
38 *server* akan menjadi masukan parameter yang dibutuhkan Socket.io.

1 Contoh penggunaan HTTP *server* akan dijelaskan sebagai berikut:

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

5
6 var app = express();
7 var httpServer = http.createServer(app);
8 var io = socketIO(httpServer);
```

Listing 3.1: Contoh penggunaan *interface* *HTTP*

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

13 Variabel *app* menginisialisasi Express.js yang akan menjadi fungsi *handler*, yang meru-  
14 pakan fungsi yang menjadi masukan suatu parameter fungsi lain. Variabel *httpServer*  
15 akan menginisialisasi objek *server* HTTP dengan menggunakan fungsi *createServer()*,  
16 dimana fungsi tersebut akan memiliki masukan yang berupa variabel *app*. Agar aplikasi  
17 dapat melakukan koneksi kepada Socket.io, maka *server* HTTP harus terintegrasi dengan  
18 Socket.io. Proses tersebut dilakukan dengan cara menginisialisasi variabel *io* dengan  
19 memberi masukan variabel *httpServer* kedalam fungsi *socketIO()*. Dengan melakukan  
20 proses ini, maka *client* dapat terkoneksi dengan Socket.io.

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

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

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

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

28
29 var app = express();
30 var httpServer = http.createServer(app);
```

Listing 3.2: Contoh penggunaan *method* *createServer(options[,requestListener])*

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

35 Salah satu kelas yang dimiliki oleh HTTP adalah sebagai berikut:

- 36 • **http.Server**

37 Kelas ini akan menciptakan objek *server* HTTP, yang akan menangani setiap

permintaan yang dilakukan oleh *client*. Salah satu *method* yang dimiliki oleh kelas ini adalah:

– **server.listen()**

*Method* ini akan memulai *server* HTTP untuk melakukan proses *listening*, yang berarti *server* akan mencari suatu *event* yang dikirimkan oleh *client* kepada *server*.

Pada pengembangan Finger For Life, *method* yang digunakan adalah *method* yang berasal dari *interface Net*<sup>2</sup> yang masih dimiliki Node.js. *Method* yang digunakan memiliki sifat yang sama dengan *method server.listen()*, namun ada tambahan masukan pada parameternya. Berikut merupakan *method* dari *interface Net*:

\* **server.listen(options[, callback])**

**Parameter:**

· **options**

Objek-objek yang dibutuhkan untuk masukan. Berikut salah satu contoh objek tersebut:

· **port**, nomor yang akan dituju oleh *server* untuk melakukan koneksi.

· **callback**

Fungsi yang menangani proses tertentu yang menjadi masukan didalam parameter fungsi lain.

**Kembalian:** Objek *server*.

Contoh penggunaan *method* ini didalam pengembangan Finger For Life adalah sebagai berikut:

```
server.listen( port , (req , res) => {
  console.log("Listening to " + port );
});
```

Listing 3.3: Contoh penggunaan *method* server.listen(options[, callback])

*Method* ini akan melakukan koneksi kepada variabel *port*, yang kemudian akan mengeksekusi fungsi *callback*. Fungsi tersebut akan mengeluarkan suatu teks pada konsol yaitu *Listening to (port)*.

(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:

Listing 3.4: contoh implementasi *method join()*

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

<sup>2</sup>[https://nodejs.org/dist/latest-v10.x/docs/api/net.html#net\\_server\\_listen\\_options\\_callback](https://nodejs.org/dist/latest-v10.x/docs/api/net.html#net_server_listen_options_callback), diakses 14 November 2018

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:

Listing 3.5: proses *export* suatu modul

```
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');
```

1 Potongan kode tersebut menunjukan bahwa nilai *ejs* akan ditetapkan kepada *view engine*. Dengan begitu, *view engine* akan mengembalikan *ejs* apabila diperlukan  
2 untuk diakses.  
3

4 • **app.use([path,] callback[, callback...])**

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

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

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

12 (c) **Response**

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

15 Pada pengembangan Finger For Life, *method* ini digunakan untuk menampilkan  
16 halaman HTML yang berada didalam parameter. Contoh implementasi dari *method*  
17 ini adalah sebagai berikut:

18 router . get ( '/ ', function ( req , res , next ) {  
19     res . render ( 'home' , { title : 'Home Pages' } );  
20 });

21 Potongan kode tersebut menunjukan bahwa apabila *client* mengakses *path* '/', yang  
22 merupakan halaman utama dari web, maka objek *res* akan menampilkan halaman  
23 HTML dari berkas yang bernama '*home*'.

24 (d) **Router**

26 • **router.METHOD()**

27 Pada pengembangan Finger For Life, *method* ini digunakan untuk mengatasi permin-  
28 taan *client* yang menggunakan salah satu *method* milik protokol HTTP. Salah satu  
29 *method* yang dipakai adalah *.get()*. Contoh implementasi dari *method* ini adalah  
30 sebagai berikut:

31 router . get ( '/ ', function ( req , res , next ) {  
32     res . render ( 'home' , { title : 'Home Pages' } );  
33 });

34 Potongan kode tersebut menunjukan bahwa objek *router* akan menjalankan perintah  
35 yang berada didalam fungsi tersebut apabila *client* mengakses *path* '/' menggunakan  
36 *method* *.get()*.

37 3. **Socket.io**

38 (a) **Server API**

39 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:

```

6   const http = require('http');
7   const socketIO = require('Socket.io');
8   const express = require('express');

9
10  var app = express();
11  var httpServer = http.createServer(app);
12  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:

```
31  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:

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

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

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

5           • **socket.id**

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

9       var socketID = socket.id

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

12           • **socket.room**

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

16       var myRoom = socket.room

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

18      Salah *method* yang dimiliki oleh kelas ini adalah sebagai berikut:

19           • **socket.join(room[, callback])**

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

26       socket.join('room305');

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

29      (b) **Client API**

31      i. **IO**

32           • **io([url][, options])**

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

36       var socket = io();

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

39      ii. **Socket**

1     ● **socket.on(eventName, callback)**

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

```
5         socket . on( ' sendEvent ' , function () {  
6             console . log ( ' An event has been sent ' );  
7         });
```

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

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

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

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

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

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

18     ● **connect**

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

```
22         socket . on ( ' connect ' , function () {  
23             console . log ( ' Connected to the server ! ' );  
24         });
```

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

28     ● **disconnect**

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

```
32         socket . on ( ' disconnect ' , function () {  
33             console . log ( ' Disconnected from the server . ' );  
34         });
```

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

37     4. Canvas API

38         (a) Animation

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

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

5             setInterval (showCountDown , 1000);

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

8     ● **clearInterval(intervalID)**

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

12             var timer = setInterval (showCountDown , 1000);

13             clearInterval (timer );

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

17     ● **requestAnimationFrame(callback)**

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

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

27             progressPlayer1 += 1;

29             ctx.restore();

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

32             if (progressPlayer1 < 40) {

33                 aniFrame = requestAnimationFrame(readyPlayerOne);

34             }

35             else {

36                 progressPlayer1 = 0;

37             }

38             }

39         *Method* *readyPlayerOne* digunakan untuk proses membuat animasi didalam perma-  
40         inan. 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();
```

1           • **CanvasRenderingContext2D.restore()**

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

6         ctx.restore();

7         5. **jQuery**

8           (a) **submit(handler)**

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

13        \$( 'form' ).submit( function(e){  
14            e.preventDefault();  
15            socket.emit( 'requestToJoin' , {  
16              id: socket.id ,  
17              room: \$('#code').val()  
18           });  
19       });

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

23           (b) **val()**

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

28        var code = \$('#code').val()

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

31           (c) **html()**

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

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

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

1       (d) **preventDefault()**

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

```
6   $('form').submit(function(e){  
7     e.preventDefault();  
8   });
```

9       *Method* ini akan mengakses elemen *form* dari HTML, dan mencegah aksi *default* dari  
 10      *method submit()* dilakukan. Aksi *default* yang dilakukan adalah melakukan *reload* pada  
 11      halaman web.

12     **6. The Content Template element**

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

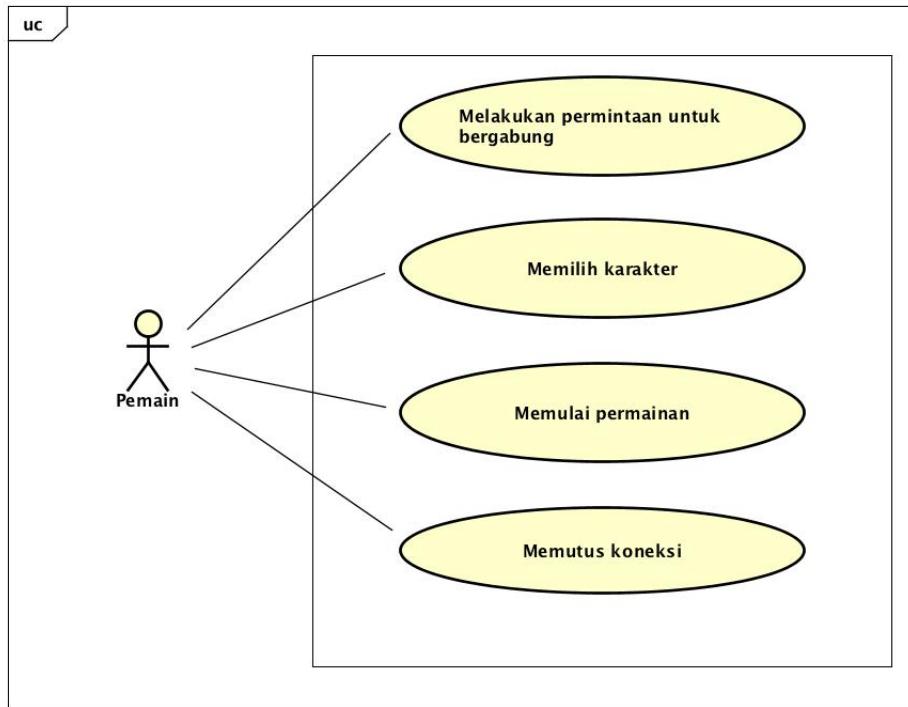
```
17 <template id="homePage">  
18   <div class="titleContainer">  
19     <div class="leftFing">  
20         
21     </div>  
22  
23     <div class="wordContainer">  
24       <div class="word">  
25         <p>FINGER <br> FOR <br> LIFE</p>  
26       </div>  
27  
28       <button id="startButton" onclick="startClicked()">START</button>  
29       <button id="joinButton" onclick="joinClicked()">JOIN</button>  
30  
31   </div>  
32  
33   <div class="rightFing">  
34       
35   </div>  
36   </div>  
37 </template>
```

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

## 1 3.4 Analisis Use Case

### 2 3.4.1 Diagram Use Case

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



Gambar 3.16: Diagram *use case* pemain

5 Berdasarkan hasil analisis pada subbab ??, akan dibentuk empat *use case* seperti berikut:

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

### 14 3.4.2 Skenario Use Case

#### 15 1. Melakukan Permintaan Untuk Bergabung

- 16 • Nama: Melakukan permintaan untuk bergabung  
17 • Aktor: Pemain

- Deskripsi: Melakukan permintaan untuk bergabung dengan *room* yang sudah tersedia
- Kondisi awal: Pemain telah membuka halaman permintaan bergabung 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

- Eksepsi: Karakter muncul dilayar *PC*.

## 3. Memainkan Permainan

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

- 1       • Karakter bergerak dari garis awal hingga akhir.

2       **4. Memutus Koneksi**

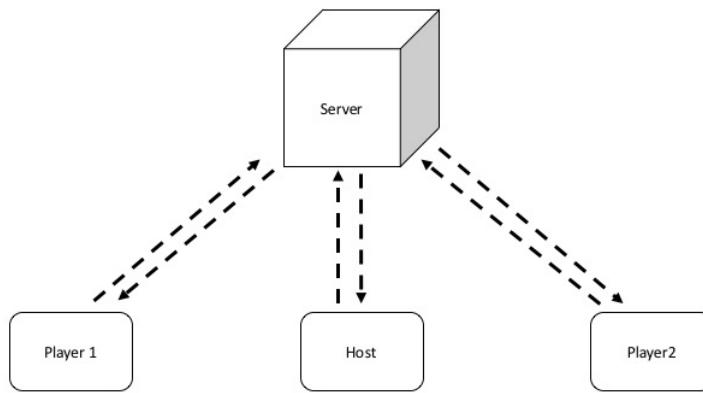
- 3       • Nama: Memutus koneksi  
 4       • Aktor: Pemain  
 5       • Deskripsi: Memutus koneksi dengan keluar dari *browser* atau menekan tombol *exit* saat  
 6       telah selesai bermain  
 7       • Kondisi awal: Pemain telah selesai bermain  
 8       • Kondisi akhir: Pemain keluar dari permainan dan koneksi terputus  
 9       • 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 <i>event</i> tombol <i>exit</i> ditekan dan memutus koneksi pemain

- 11      • Eksepsi: Seluruh *client* keluar dari permainan.

12      **3.5 Analisis Arsitektur Finger For Life**

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



Gambar 3.17: Arsitektur Finger For Life

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

19     Finger For Life dibangun diatas Node.js. Seluruh berkas serta direktori yang dibutuhkan untuk  
 20   membangun aplikasi permainan ini diatur dengan menggunakan Express.js. Proses pengiriman

1 dan penerimaan data secara *realtime* dilakukan menggunakan Socket.io. Dalam permainan ini,  
 2 animasi akan dilakukan dengan menggunakan Canvas API. Seluruh proses memodifikasi elemen  
 3 HTML akan dilakukan dengan menggunakan jQuery. Halaman-halaman yang dibutuhkan untuk  
 4 ditampilkan kelayar akan diatur dengan menggunakan The Content Template element.

### 5 **3.6 Analisis Socket.io**

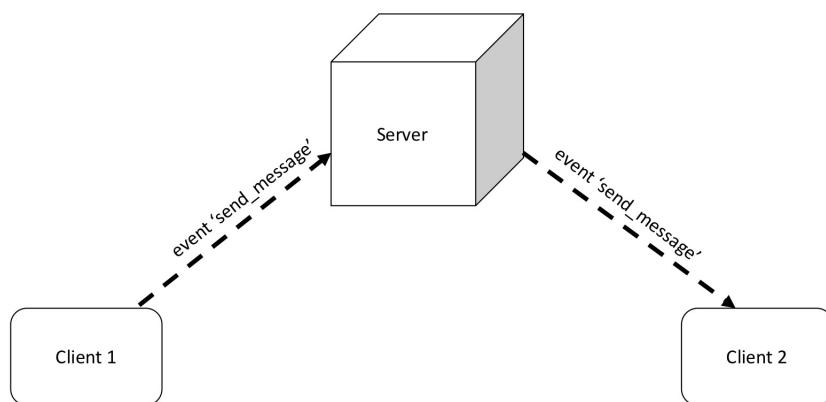
6 Pada pengembangan Finger For Life digunakan banyak fitur yang telah disediakan oleh Socket.io.  
 7 Teknologi ini digunakan untuk proses sinkronisasi antara *smartphone* dengan *PC* dan komunikasi  
 8 secara *real-time* antara *client* dan *server*. Seperti yang sudah dijelaskan pada subbab ??, ada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

11 Subbab-subbab ini akan menjelaskan bagaimana teknologi Socket.io digunakan dalam imple-  
 12 mentasi Finger For Life pada tahap-tahap yang harus dilakukan.

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

14 Pada aplikasi permainan berbasis web ini, akan banyak dibutuhkan proses komunikasi antara  
 15 sesama *client* maupun dengan *server*. Komunikasi yang dilakukan harus secara *real-time*, agar  
 16 interaksi didalam permainan dapat berjalan pada saat permainan dimainkan.

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



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

23 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 parameter *eventName*. *Method* ini yang akan menangkap suatu *event* yang telah dipancarkan kedalam *stream*. Setelah *event* tersebut ditangkap, maka fungsi *callback* akan dieksekusi.

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

```

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 ;
});

```

Listing 3.6: potongan kode pada bagian *client*

Potongan kode ini menunjukkan 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*.

```

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 ' );
    }
});

```

Listing 3.7: potongan kode pada bagian *server*

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*.

```
8   socket .on( 'requestAccepted' , function (msg){  
9       showMessage (msg);  
10  } );
```

Listing 3.8: proses *event* diterima

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 terkoneksi dengan Socket.io. Proses melakukan koneksi terhadap Socket.io dilakukan pada saat halaman web mengakses halaman permintaan bergabung. 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:

```
28  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 permintaan bergabung, maka kedua gawai tersebut telah terkoneksi dengan Socket.io. Walaupun sudah terkoneksi, 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:

```

1   function getRandInt(){
2     text = Math.floor(Math.random() * (999999 - 111111)) + 111111;
3     document.getElementById("roomId").innerHTML = text;
4   }

```

Listing 3.9: proses membangkitkan kode *room*

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:

```

9   function getRandInt(){
10     text = Math.floor(Math.random() * (999999 - 111111)) + 111111;
11     document.getElementById("roomId").innerHTML = text;
12
13     var roomString = text.toString();
14
15     socket.emit('hostJoinRoom', {
16       id: socket.id,
17       room: roomString
18     });
19   }

```

Listing 3.10: proses *host* bergabung kedalam room

*PC* akan memancarkan *event hostJoinRoom* 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:

```

24   socket.on('hostJoinRoom', (msg) => {
25     socket.join(msg.room);
26     users.removeUser(msg.id);
27     users.addUser(msg.id, msg.room);
28   });

```

Listing 3.11: server menerima *event* dari *host*

*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:

```

1   function requestToJoin(){
2       $( 'form' ).submit( function(e){
3           e.preventDefault();
4           socket.emit( 'requestToJoin' , {
5               id: socket.id ,
6               room: $('#code').val()
7           });
8       });
9   }
10
11 }
```

Listing 3.12: proses permintaan bergabung pada *client*

*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 verifikasi. Proses tersebut akan dilakukan seperti berikut:

```

18 socket.on( 'requestToJoin' , function(msg){
19     var check = users.isRoomExist(msg.room);
20
21     if (check === 1) {
22         socket.join(msg.room);
23         users.removeUser(msg.id);
24         users.addUser(msg.id , msg.room);
25
26         io.to(user.room).emit( 'requestAccepted' , msg);
27         socket.emit( 'joinSucceed' , 'Welcome to the game :)');
28
29     } else {
30         socket.emit( 'joinRejected' , 'The room is not exist');
31     }
32 })
```

Listing 3.13: Proses verifikasi

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.

1

## BAB 4

2

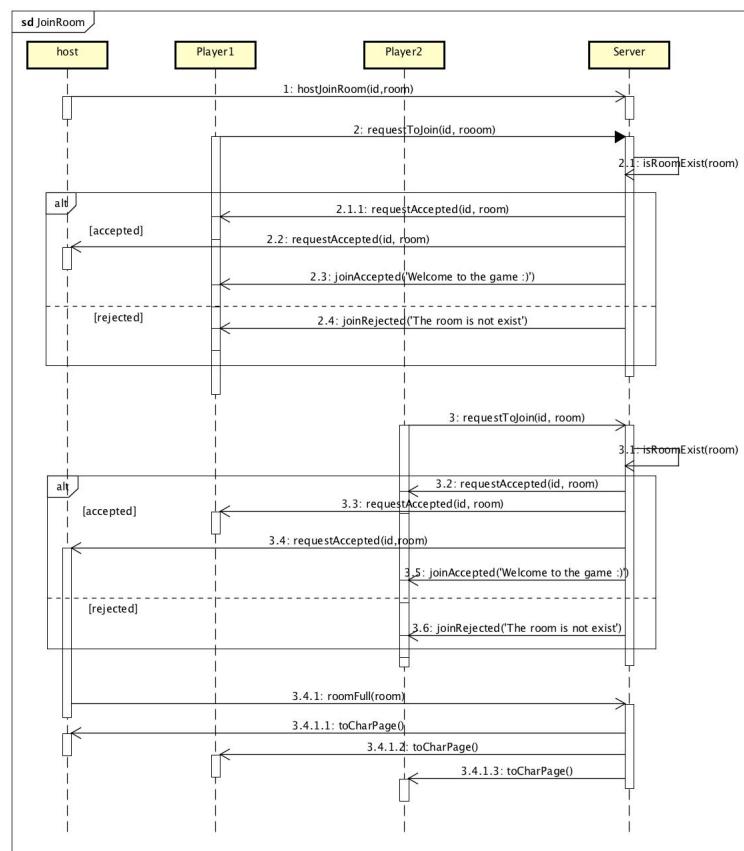
### PERANCANGAN

- 3 Pada bab ini akan dijelaskan mengenai perancangan aplikasi yang akan dibangun meliputi *sequence* diagram, antarmuka, dan struktur direktori.

5 **4.1 Perancangan *Sequence* Diagram**

- 6 Pada *sequence* diagram akan dibahas mengenai jalannya koneksi Socket.io dari awal koneksi mulai 7 tersambung hingga koneksi terputus. *Sequence* diagram meliputi *sequence* permintaan bergabung, 8 *sequence* memilih karakter, *sequence* memulai permainan, dan *sequence* mengakhiri permainan.

9 **4.1.1 *Sequence* Permintaan Bergabung**



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

1 Pada awal permainan, *client* pertama yang melakukan koneksi pada Socket.io adalah *PC*, yang  
2 berperan sebagai *host* dalam permainan seperti yang dijelaskan pada Gambar 4.1.*Host* akan  
3 menyediakan suatu kode yang berguna sebagai *room* untuk kedua pemain yang akan bergabung  
4 dengan melakukan koneksi ke *server*. *Room* yang disediakan hanya akan menerima tiga *client* saja,  
5 yaitu *host*, *player1*, dan *player2*.

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

- 9 • **id**, identifikasi unik yang dimiliki masing-masing *client* yang terkoneksi dengan Socket.io.
- 10 • **room**, suatu *String* yang menandakan ruangan dimana *client* hanya akan melakukan komuni-  
11 kasi didalam ruangan tersebut.

12 Data-data tersebut akan dikirimkan ke *server* dan akan dimasukan kedalam suatu *array* yang  
13 diproses oleh kelas *Users*. Kelas ini memiliki *array* dengan nama *userArray* yang menyimpan  
14 seluruh *client* yang telah terkoneksi dengan Socket.io. Kelas *Users* akan bertanggung jawab dalam  
15 menyambung koneksi maupun memutus koneksi antara *client* dan *server*.

16 Setelah *host* terkoneksi dengan Socket.io, maka suatu *room* sudah tersedia dan para pemain dapat  
17 bergabung kedalam *room* tersebut. Untuk dapat mulai bermain, para pemain harus memasukan  
18 kode *room* yang disediakan di halaman *host* kedalam *form* yang ada dihalaman *browser* pada  
19 *smartphone*. Pemain akan mengirimkan *event* **requestToJoin(id, room)** pada *server*. Data yang  
20 dikirimkan oleh pemain sama dengan data yang dikirimkan oleh *host*.

21 Setelah *event* tersebut diterima oleh *server*, maka akan dilakukan pengecekan apakah pemain  
22 dapat bergabung atau tidak kedalam *room*. Pengecekan akan dilakukan didalam kelas *Users*.  
23 Pengecekan tersebut dilakukan dengan menggunakan *method* **isRoomExist(room)**. Beberapa  
24 pengecekan yang dilakukan *method* ini akan melakukan proses seperti berikut:

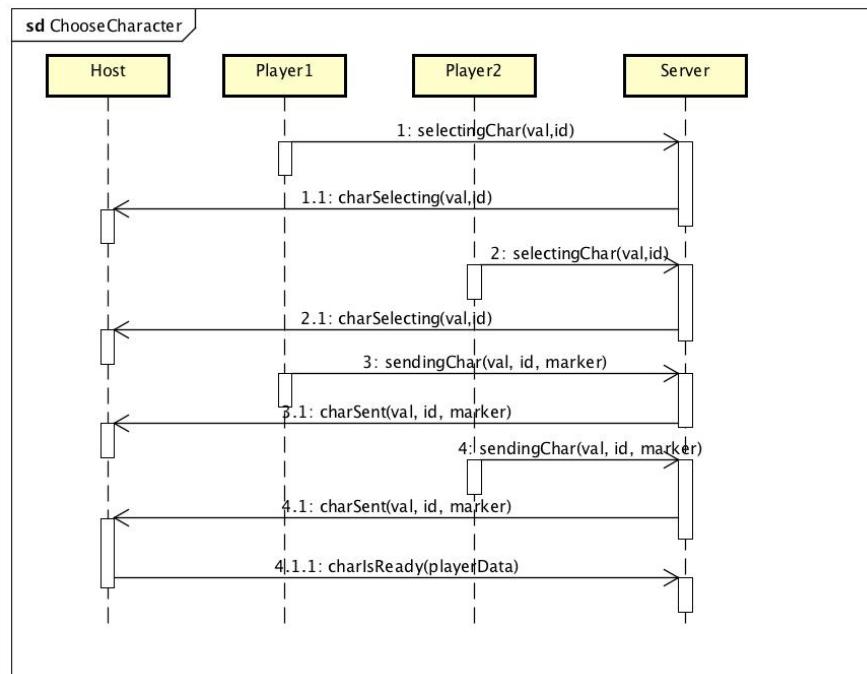
- 25 1. Memeriksa apakah data kode *room* didalam *userArray* ada yang sesuai dengan parameter  
26 *room*.
- 27 2. Memeriksa apakah jumlah *client* yang ada didalam *room* yang dimaksud sudah lebih dari tiga  
28 atau belum.

29 Apabila kedua hal diatas terpenuhi, maka pemain akan terkoneksi ke Socket.io dan bergabung  
30 kedalam *room*. *Server* akan mengirimkan *event* **requestAccepted(id, room)** kepada seluruh  
31 *client* yang berada didalam *room* tersebut. *Event* tersebut akan ditangkap oleh *host* yang berfungsi  
32 untuk mencatat sudah ada berapa pemain yang berhasil masuk kedalam *room*. *Server* pun akan  
33 mengirimkan *event* **joinAccepted('Welcome to the game :)')** kepada pemain yang berhasil  
34 bergabung kedalam *room*. *Event* tersebut berfungsi untuk memberikan *feedback* kepada pemain  
35 bahwa pemain telah berhasil bergabung kedalam permainan.

36 Apabila kedua hal diatas tidak terpenuhi, maka pemain tidak dapat terkoneksi ke Socket.io.  
37 *Server* akan mengirimkan *event* **joinRejected('The room is not exist')** kepada pemain. *Event*  
38 tersebut berfungsi sebagai *feedback* kepada pemain yang tidak dapat bergabung. Pemain pertama  
39 yang berhasil bergabung akan berperan sebagai *Player1*. Setelah satu pemain bergabung, maka

1 *server* akan menunggu untuk pemain kedua agar permainan dapat dimulai. Pemain kedua akan  
 2 melakukan hal yang sama dengan pemain pertama untuk dapat bergabung kedalam room.  
 3 Setelah *room* berisi tiga *client*, maka *host* akan mengirimkan *event roomFull(room)* kepada  
 4 *server*. *Event* tersebut menandakan bahwa *room* sudah tidak dapat menerima *client* yang akan  
 5 bergabung. Setelah *event* tersebut diterima oleh *server*, maka *event toCharPage()* akan dikirimkan  
 6 oleh *server* kepada seluruh *client* yang ada didalam *room* tersebut. *Event* ini berfungsi untuk  
 7 mengganti halaman saat ini ke halaman selanjutnya.

#### 8 4.1.2 Sequence Memilih Karakter



Gambar 4.2: Proses memilih karakter.

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

15 • **val**, identifikasi unik yang dimiliki oleh masing-masing karakter.

16 • **id**, identifikasi unik yang dimiliki masing-masing *client* yang terkoneksi dengan Socket.io.

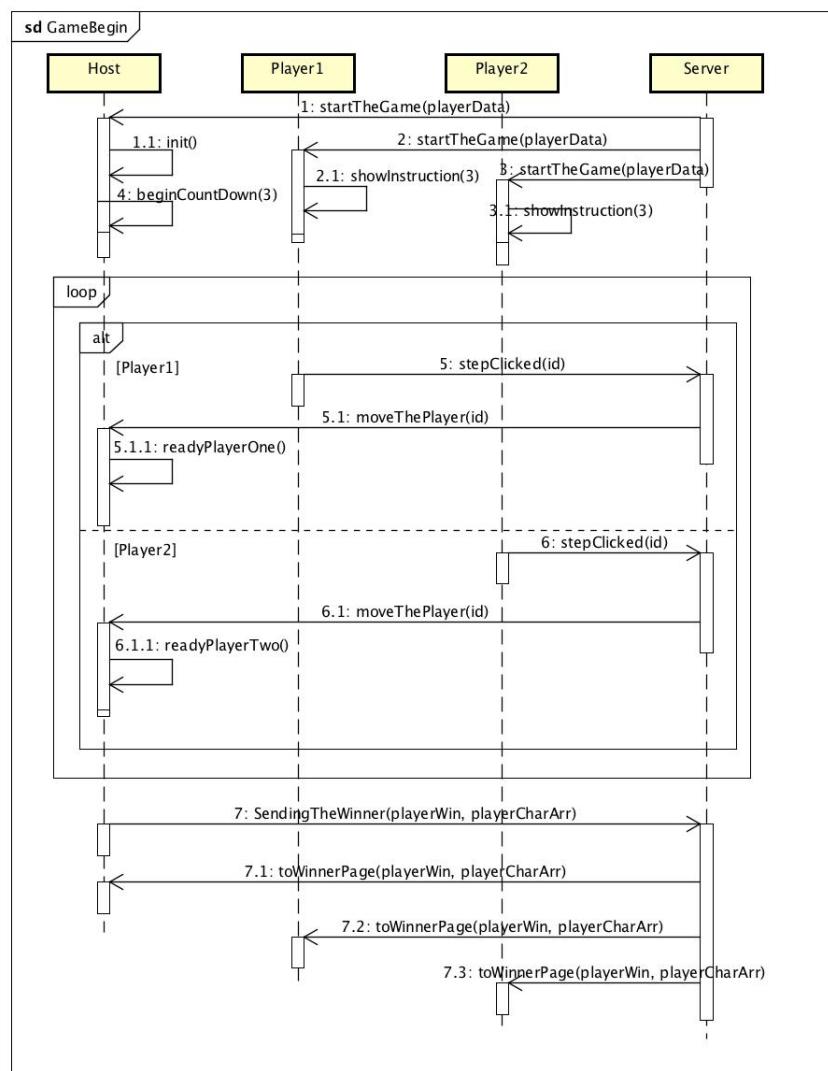
17 Setelah *event* tersebut diterima oleh *server*, maka server akan mengirimkan *event charSele-*  
 18 *cting(val, id)* kepada *host*. *Event* tersebut berfungsi untuk menampilkan karakter yang dipilih  
 19 oleh pemain dihalaman *PC*.

20 Apabila pemain akan menetapkan karakter yang telah ditampilkan dihalaman *PC*, maka *pemain*  
 21 akan mengirimkan *event sendingChar(val,id,marker)* kepada *server*. Data-data tersebut sama

1 seperti yang dikirimkan oleh *event* sebelumnya, hanya saja ada tambahan data **marker** pada  
 2 parameter. Data tambahan tersebut berfungsi sebagai penanda bahwa sudah ada satu pemain yang  
 3 telah menetapkan karakter yang akan dimainkan. Setelah *server* menerima *event* tersebut, maka  
 4 *server* akan meneruskan data-data yang sudah diterima kepada *host* dengan mengirimkan *event*  
 5 **charSent(val,id,marker)**. Apabila *event* tersebut telah diterima oleh *host*, maka karakter telah  
 6 dimiliki oleh pemain yang menetapkannya.

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

#### 12 4.1.3 Sequence Memulai Permainan



Gambar 4.3: Proses memulai permainan.

1 Pada tahap ini, *server* akan memancarkan *event startTheGame(playerData)*. Seperti yang  
2 dijelaskan pada Gambar 4.3, data yang dikirimkan merupakan data yang didapatkan dari *event*  
3 sebelumnya. *Event* tersebut akan diterima oleh seluruh *client* yang berada didalam *room* tertentu,  
4 yang kemudian akan memulai permainannya. Untuk memulai permainan, *host* akan mengeksekusi  
5 *method init()* dan **beginCountDown(3)**. *Method init()* berfungsi untuk mulai menggambar  
6 lintasan lari di halaman *PC* yang akan menjadi tempat para karakter dimainkan. *Method be-*  
7 **ginCountDown(3)** berfungsi untuk menampilkan hitungan mundur selama tiga detik sebelum  
8 permainan dapat dimulai.

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

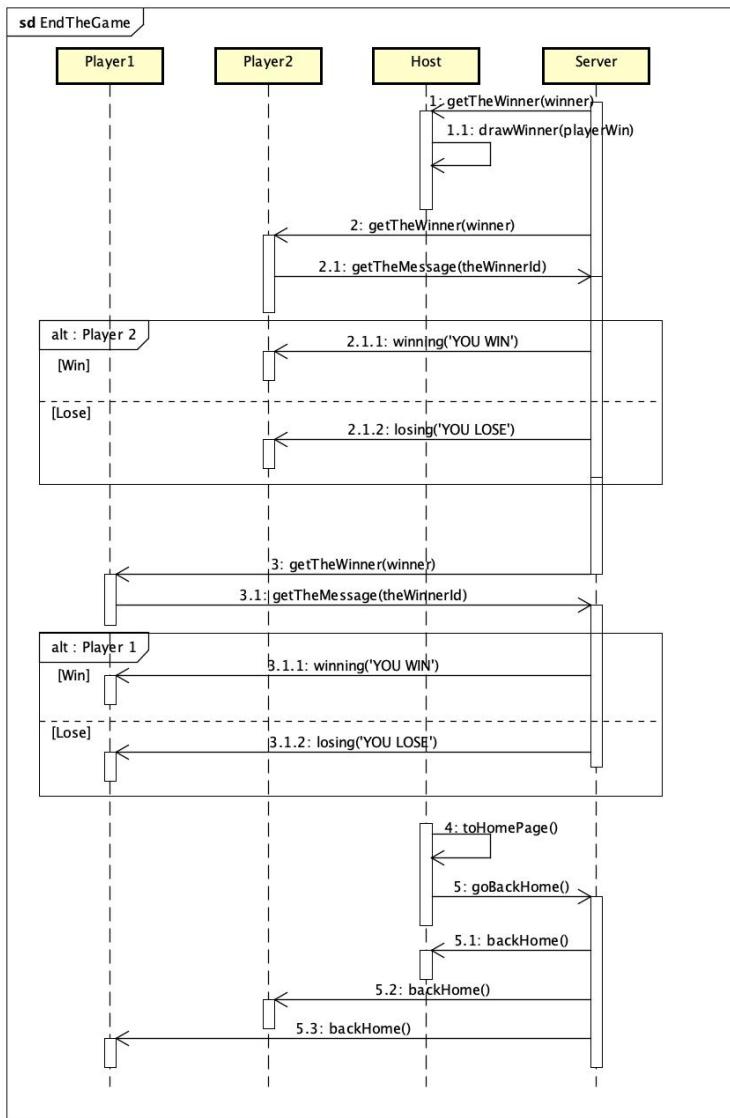
13 Permainan dilakukan dengan cara menekan tombol berbentuk telapak kaki yang ada pada halam-  
14 an *smartphone*. Apabila pemain menekan tombol, maka *event stepClicked(id)* akan dipancarkan.  
15 *Server* akan menerima *event* tersebut, yang kemudian akan mengirim *event moveThePlayer(id)*  
16 kepada *host*. Setelah *event* tersebut diterima oleh *host*, maka *host* akan menjalankan *method*  
17 tertentu untuk menggerakan karakter milik pemain. Apabila *Player1* yang menekan tombol, maka  
18 *method readyPlayerOne()* akan dieksekusi oleh *host*. Apabila *Player2*, maka **readyPlayerTwo()**  
19 yang akan dieksekusi. Kedua *Method* tersebut berfungsi untuk memindahkan posisi karakter milik  
20 pemain dari posisi semula hingga posisi tertentu. Permainan akan berakhir apabila ada karakter  
21 yang menyentuh garis akhir pertama kali.

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

- 25 • **playerWin**, angka *integer* yang menandakan pemain nomor berapa yang memenangkan  
26 permainan.
- 27 • **playerCharArr**, *array* yang menyimpan karakter dari masing-masing pemain.

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

#### 1 4.1.4 Sequence Mengakhiri Permainan



Gambar 4.4: Menampilkan para pemain yang telah selesai bermain.

- 2 Pada tahap ini, *server* akan memancarkan *event getTheWinner(winner)* dengan data-data pada parameter yang diterima dari *event* sebelumnya. Seperti yang dijelaskan pada Gambar 4.4, *host* akan menerima *event* tersebut dan akan mengeksekusi *method drawWinner(playerWin)*. *Method* tersebut berfungsi untuk menampilkan karakter milik para pemain diatas gambar podium yang telah selesai bermain. Parameter **playerWin** berfungsi untuk mengetahui pemain nomor berapa yang menduduki posisi pertama. Dengan begitu, pemenang akan ditampilkan pada podium atas, sedangkan pihak yang kalah di podium bawah.
- 9     *Event getTheWinner(winner)* pun akan diterima oleh para pemain, yang kemudia para pemain akan memancarkan *event getTheMessage(theWinnerId)* kepada *server*. *Server* tersebut akan memberi informasi kepada *server socket.id* milik pemain yang memenangkan permainan. Setelah *server* menerima *event* tersebut, maka *server* akan mengirim *event* kembali kepada pemain. 13 *Server* akan memancarkan *event winning('YOU WIN')* kepada pihak yang menang, dimana

- 1 *event* tersebut akan menampilkan pesan '**YOU WIN**' dilayar *smartphone*. Sedangkan untuk pihak
- 2 yang kalah, *server* akan memancarkan *event losing*('**YOU LOSE**').

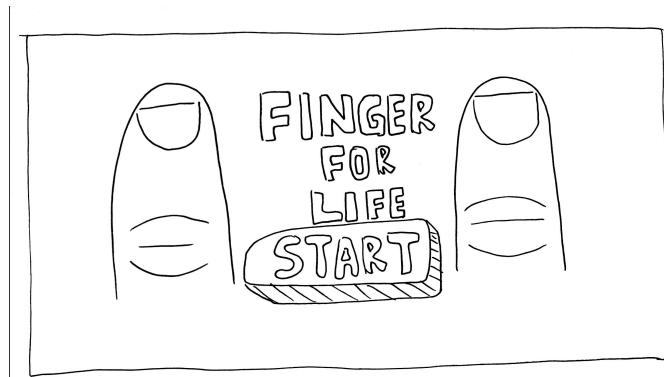
### **4.2 Perancangan Antarmuka**

- 4 Antarmuka yang dirancang terbagi menjadi dua bagian, yaitu antarmuka yang ada di *PC* dan
- 5 *smartphone*. Subbab berikut akan

- 6 1. Antarmuka halaman utama

#### **PC**

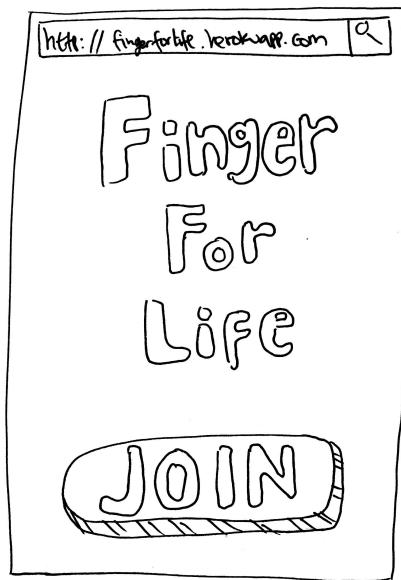
Halaman ini merupakan halaman utama yang pertama kali dituju oleh pengguna yang menggunakan *PC*. Komponen halaman ini terdiri dari dua buah gambar jari yang melambangkan permainan, teks yang menunjukan nama permainan yaitu *Finger For Life*, dan tombol *start* yang digunakan untuk memulai permainan.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



Gambar 4.5: Halaman pada *PC* yang menunjukan halaman utama 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 menunjukan nama permainan yaitu *Finger For Life*, dan tombol *join* yang digunakan untuk memulai permainan. Rancangan antarmuka halaman utama dapat dilihat pada Gambar 4.6.

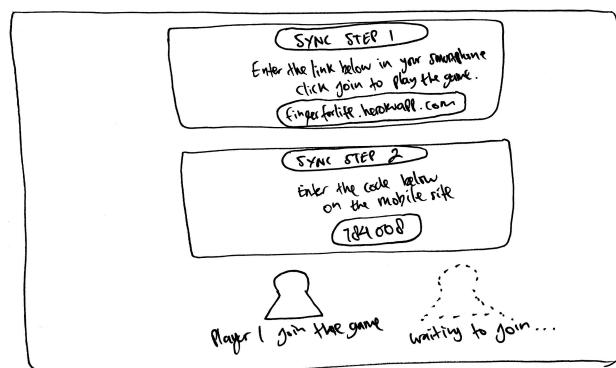


Gambar 4.6: Halaman pada *smartphone* yang menunjukan halaman utama saat *client* mengakses alamat web.

<sup>1</sup> 2. Antarmuka halaman Permintaan Bergabung

<sup>2</sup> **PC**

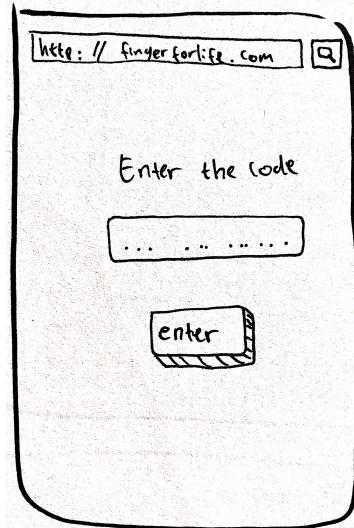
<sup>3</sup> Halaman ini menampilkan suatu perintah yang dapat dilakukan oleh pengguna apabila akan <sup>4</sup> bergabung kedalam permainan. Halaman ini menyediakan suatu kode untuk para pemain <sup>5</sup> dimana kode tersebut akan berfungsi sebagai suatu *room* bagi para pemain. Pada halaman ini <sup>6</sup> akan dilakukan proses sinkronisasi antara *smartphone* dan *PC*, dan akan dilakukan pengecekan <sup>7</sup> apakah berhasil bergabung dalam permainan atau tidak. Apabila berhasil, maka akan muncul <sup>8</sup> suatu teks yang menunjukkan seorang pemain telah berhasil bergabung. Apabila tidak, maka <sup>9</sup> tidak akan menampilkan apapun dan halaman tidak akan menuju ke halaman selanjutnya <sup>10</sup> sebelum ada dua pemain yang berhasil bergabung. Rancangan antarmuka halaman permintaan <sup>11</sup> bergabung pada *PC* dapat dilihat pada Gambar 4.7.



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

<sup>12</sup> **Smartphone**

Halaman ini berfungsi untuk melakukan permintaan bergabung kedalam permainan. Komponen halaman ini terdiri dari kolom untuk mengisi kode, dan tombol *send*. *client* harus mengisi kode yang disediakan dihalaman *PC* kedalam kolom. Setelah mengisi kode, maka *client* harus menekan tombol *send* untuk mengirim kode tersebut kepada *server* untuk dilakukan pengecekan. Rancangan antarmuka halaman permintaan bergabung pada *smartphone* dapat dilihat pada Gambar 4.8.

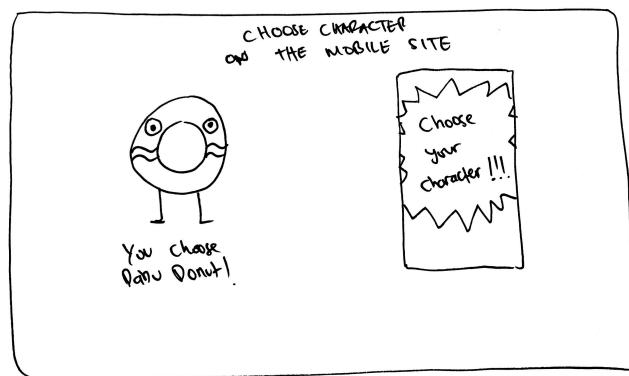


Gambar 4.8: Halaman pada *smartphone* yang menampilkan kolom untuk mengisi kode.

### 3. Antarmuka halaman Memilih Karakter

#### **PC**

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

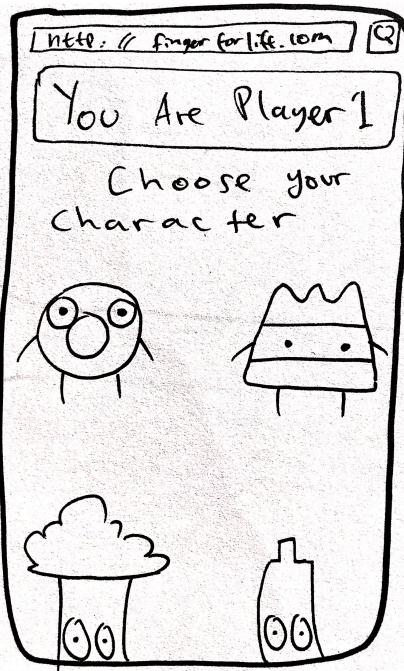


Gambar 4.9: Halaman pada *PC* yang menampilkan karakter yang telah ditetapkan oleh pemain.

#### **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

memilih karakter pada *smartphone* dapat dilihat pada Gambar 4.10.

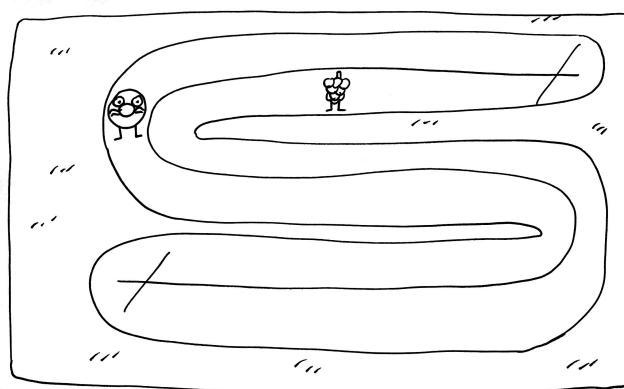


Gambar 4.10: Halaman pada *smartphone* yang menampilkan daftar karakter yang dapat dipilih.

#### 4. Antarmuka halaman Memulai Permainan

##### **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 memulai permainan 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* akan mengirimkan nilai dari karakter tertentu yang dipilih. Nilai tersebut dikirimkan melalui *event* Socket.io yang dipancarkan oleh *client* dengan nama *sendingChar*.

1        Server akan menangkap *event* tersebut, kemudian akan memancarkan *event* dengan  
2        nama *startTheGame*, dimana *event* tersebut akan ditangkap oleh *host* untuk memulai  
3        permainan.

- 4        • *Host* akan menggambar karakter pada layar *PC* setelah mendapatkan data-data dari  
5        *event startTheGame*.

6        Untuk menampilkan karakter kelayar *PC* saat proses animasi, dibutuhkan teknologi Canvas  
7        API untuk menggambar karakter agar ditampilkan kelayar *PC*. Canvas API akan menciptakan  
8        objek *canvas*, yang akan digunakan sebagai wadah dalam menggambar seluruh elemen yang  
9        diperlukan untuk permainan. Pada aplikasi permainan ini, objek *canvas* akan dibuat dengan  
10      ukuran panjang 900 piksel, dan lebar 600 piksel. Untuk memulai proses animasi, ada beberapa  
11      langkah yang harus dilakukan. Langkah-langkah tersebut dijelaskan sebagai berikut:

12      (a) **Menghapus isi *canvas***

13      Untuk memulai animasi, langkah pertama yang harus dilakukan adalah menghapus  
14      seluruh elemen yang berada di dalam *canvas*. Langkah tersebut dilakukan untuk meng-  
15      hilangkan bentuk atau elemen lain yang sudah digambar sebelumnya didalam *canvas*,  
16      sehingga dapat menggambar suatu *frame*, yang merupakan tempat dimana animasi  
17      akan berlangsung. Proses menghapus akan dilakukan dengan menggunakan *method*  
18      *clearRect(0, 0, 900, 600)*. Dengan *method* tersebut, seluruh elemen yang ada didalam  
19      *canvas* dengan ukuran panjang 900 piksel dan lebar 600 piksel akan dihilangkan.

20      (b) **Menyimpan *state* milik *canvas***

21      *State* adalah kondisi saat ini dari objek *canvas*. Apabila akan dilakukan perubahan seperti  
22      merubah warna bentuk, transformasi bentuk, atau hal lainnya yang akan menyebabkan  
23      perubahan pada *state* dari *canvas*, maka *state* harus disimpan setiap ada perubahan.  
24      Proses penyimpanan *state* akan dilakukan dengan menggunakan *method* *save()*.

25      (c) **Menggambar elemen untuk animasi**

26      Pada langkah ini, elemen yang akan diproses untuk animasi akan digambar kedalam  
27      *canvas*. Elemen yang digambar dapat berupa bentuk bangunan, gambar, maupun  
28      tulisan. Proses menggambar kedalam *canvas* akan dilakukan dengan menggunakan  
29      *method* *drawImage()*.

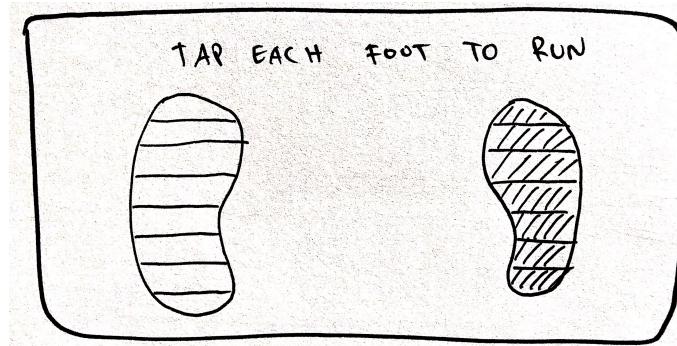
30      (d) **Mengembalikan *state* dari *canvas***

31      Beberapa *state* dari *canvas* yang sudah disimpan, harus dikembalikan seluruhnya agar  
32      kembali ke *state* pertama sebelum ada perubahan apapun. Hal tersebut dilakukan sebelum  
33      menggambar *frame* baru untuk proses animasi selanjutnya. Proses mengembalikan *state*  
34      dari *canvas* akan dilakukan dengan menggunakan *method* *restore()*. Proses animasi akan  
35      diulang kembali mulai dari langkah pertama.

36      **Smartphone**

37      Halaman ini berfungsi sebagai pengendali untuk para pemain. Komponen halaman ini terdiri  
38      dari dua buah gambar telapak kaki yang berfungsi sebagai tombol. Pemain dapat menekan  
39      tombol berulang kali untuk menggerakan karakter yang ada pada halaman memulai permainan

di *PC*. Rancangan antarmuka halaman memulai permainan pada *smartphone* dapat dilihat pada Gambar 4.12.

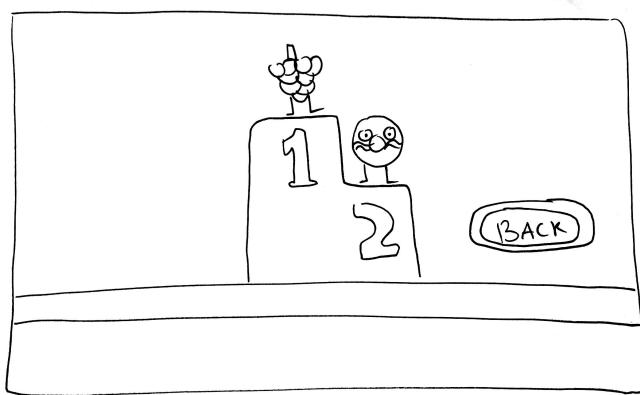


Gambar 4.12: Halaman pada *smartphone* yang menampilkan telapak kaki yang berfungsi sebagai pengendali.

### 5. Antarmuka halaman Mengakhiri Permainan

#### **PC**

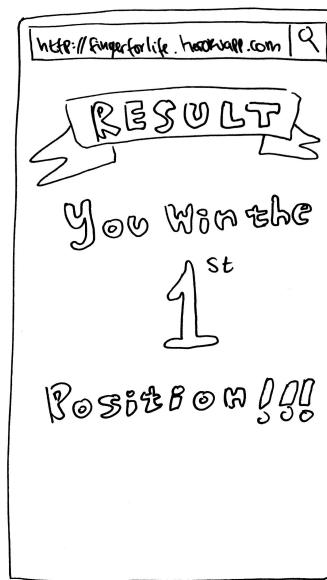
Halaman ini menampilkan para pemenang yang telah berhasil menyelesaikan permainan. Setelah pemain selesai bermain, pemain dapat menekan tombol *exit* untuk kembali ke halaman utama dan mengakhiri permainan. Rancangan antarmuka halaman mengakhiri permainan pada *PC* dapat dilihat pada Gambar 4.13.



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

#### **Smartphone**

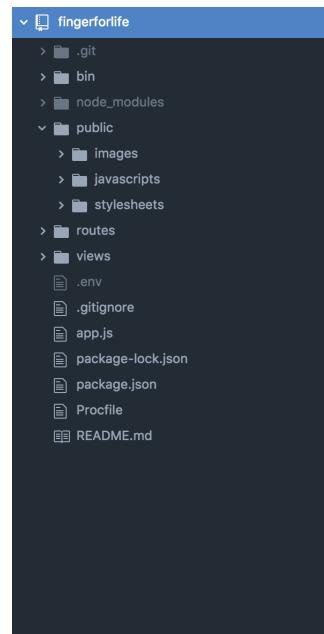
Halaman ini menampilkan teks yang menandakan bahwa permainan telah selesai. Rancangan antarmuka halaman mengakhiri permainan pada *smartphone* dapat dilihat pada Gambar 4.14.



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

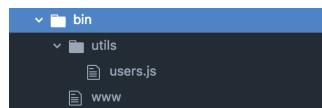
### **1 4.3 Perancangan Struktur Direktori**

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



Gambar 4.15: Seluruh direktori dan berkas yang digunakan didalam pengembangan aplikasi Finger For Life

- 5 1. **Direktori bin**



Gambar 4.16: Isi direktori bin.

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

(a) Direktori **utils**

Direktori ini menyimpan berkas yang membantu proses berjalannya koneksi untuk *server*.

Isi dari direktori ini adalah sebagai berikut:

- Berkas **users.js**

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

Kelas *Users* memiliki konstruktor yang menginisialisasi variabel *array* dengan nama *userArray*.

Konstruktor yang dimiliki oleh berkas ini adalah sebagai berikut:

```
constructor(){
    this.userArray = [];
}
```

Listing 4.1: konstruktor milik kelas *Users*

Variabel *userArray* bertanggung jawab untuk menyimpan seluruh *client* yang terkoneksi ke Socket.io.

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

- **addUser(id, room)**

*Method* ini akan menambahkan pengguna kedalam *array* yang bernama *userArray*.

**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 kode *room* milik *client* yang ada didalam *userArray*.

**Parameter:**

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

- **Kembalian:** *Integer*

- **getUser(id)**

*Method* ini akan mendapatkan pengguna dengan parameter *id* yang cocok dengan

1            *id* yang ada didalam *userArray*.

2            **Parameter:**

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

4            **Kembalian:** Objek Users

5            – **removeUser(id)**

6            *Method* ini akan menghapus pengguna dengan *id* tertentu yang ada didalam  
7            *userArray*.

8            **Parameter:**

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

10          **Kembalian:** Objek Users

11          – **getUserList(room)**

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

14          **Parameter:**

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

16          **Kembalian:** Array objek *Users*

17          (b) **Berkas www**

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

20          Properti yang dimiliki oleh berkas ini akan dijelaskan sebagai berikut:

- 21          • **app**, variabel yang mendapatkan seluruh fitur dari berkas lain yang bernama *app.js*.
- 22          • **http**, variabel yang mendapatkan seluruh fitur dari modul *HTTP*.
- 23          • **socketIO**, variabel yang mendapatkan seluruh fitur dari modul Socket.io.
- 24          • **Users**, variabel yang mendapatkan seluruh fitur dari kelas *Users* dari berkas *users.js*.
- 25          • **port**, nilai *port* yang akan dituju oleh *server*.
- 26          • **server**, objek HTTP *server* yang memiliki argumen variabel *app* yang berfungsi  
27          sebagai *handler* apabila ada koneksi yang masuk.
- 28          • **io**, *server* yang akan menangani koneksi Socket.io antara *client* dan *server*.

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

- 30          • **normalizePort(val)**

31          *Method* ini berfungsi untuk mengubah bentuk parameter *val* menjadi *String*, *Integer*,  
32          atau *Boolean*.

33          **Parameter:**

- 34          – **val**, parameter *port*

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

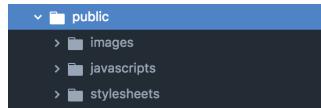
- 36          • **app.set('port', port)**

37          *Method* ini akan menetapkan nilai yang dimiliki oleh variabel *port*, kepada nama  
38          '*port*' yang digunakan untuk mengatur perilaku *server*.

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

- 1     • **io.on('connection', (socket) => {})**  
2         Berfungsi untuk menangani suatu koneksi Socket.io yang baru tersambung.
- 3     • **socket.on('hostJoinRoom', (msg) => {})**  
4         Berfungsi untuk menangkap *event hostJoinRoom* yang dipancarkan oleh *host* pada  
5         saat akan masuk kedalam room.
- 6     • **socket.on('sendingTheWinner', (winner) => {})**  
7         Berfungsi untuk menangkap *event sendingTheWinner* yang dipancarkan oleh  
8         *client* setelah ada pemenang permainan.
- 9     • **socket.on('getTheMessage', (message) => {})**  
10         Berfungsi untuk menangkap *event getTheMessage* yang dipancarkan oleh *client*  
11         pada saat ada salah satu pemain yang telah menyentuh garis akhir dari permainan.
- 12     • **socket.on('goBackHome', (msg) => {})**  
13         Berfungsi untuk menangkap *event goBackHome* yang dipancarkan oleh *client* pada  
14         saat pemain menekan tombol *exit* untuk kembali kehalaman utama.
- 15     • **socket.on('ping', (message) => {})**  
16         Berfungsi untuk menangkap *event ping* yang dipancarkan oleh *client* untuk menghi-  
17         tung *latency* pada saat pemain menekan tombol telapak kaki pada layar *smartphone*,  
18         dan karakter digerakan pada layar *PC*.
- 19     • **socket.emit('pong', message)**  
20         Berfungsi untuk memancarkan *event pong* yang digunakan untuk memberi informasi  
21         kepada *host* agar menghitung *latency* pada saat *client* menekan tombol telapak kaki  
22         pada layar *smartphone*
- 23     • **socket.on('roomFull', (msg) => {})**  
24         Berfungsi untuk menangkap *event roomFull* yang dipancarkan oleh *client* pada  
25         saat *room* telah terisi penuh.
- 26     • **socket.on('selectingChar', (msg) => {})**  
27         Berfungsi untuk menangkap *event selectingChar* yang dipancarkan oleh *client*  
28         pada saat karakter telah dipilih oleh pemain untuk menampilkan karakter kelayar  
29         *PC*.
- 30     • **socket.on('sendingChar', (msg) => {})**  
31         Berfungsi untuk menangkap *event sendingChar* yang dipancarkan oleh *client* pada  
32         saat karakter telah ditetapkan oleh pemain.
- 33     • **socket.on('charIsReady', (msg) => {})**  
34         Berfungsi untuk menangkap *event charIsReady* yang dipancarkan oleh *client* pada  
35         saat kedua karakter telah ditetapkan oleh kedua pemain.
- 36     • **socket.on('stepClicked', (message) => {})**  
37         Berfungsi untuk menangkap *event stepClicked* yang dipancarkan oleh *client* pada  
38         saat tombol telapak kaki ditekan oleh pemain.
- 39     • **socket.on('requestToJoin', (msg) => {})**  
40         Berfungsi untuk menangkap *event requestToJoin* yang dipancarkan oleh *client*  
41         pada saat melakukan proses permintaan bergabung.

1      2. Direktori public



Gambar 4.17: Isi dari folder public.

3 Direktori ini menyimpan seluruh berkas yang memiliki sifat statis, yang artinya tidak perlu  
4 dibangkitkan, dimodifikasi, atau diubah terlebih dahulu sebelum diberikan kepada *client*.  
5 Berikut akan dijelaskan isi dari direktori ini:

6      (a) **Dirketori images**

7      Direktori ini berisi gambar-gambar yang dibutuhkan untuk ditampilkan dibeberapa  
8      halaman yang terdapat didalam web. Berikut merupakan daftar isi dari gambar-gambar  
9      yang ada didalam direktori ini:

- 10     • **BrocoDude.png** gambar karakter brokoli dengan ukuran besar, yaitu lebar 1746  
11     dan tinggi 2454 piksel.
- 12     • **brocoDudeTiny.png** gambar karakter brokoli dengan ukuran kecil, yaitu lebar 65  
13     dan tinggi 92 piksel.
- 14     • **DabuDonut.png** gambar karakter donat dengan ukuran besar, yaitu lebar 1746  
15     dan tinggi 2454 piksel.
- 16     • **dabuDonut.png** gambar karakter donat dengan ukuran kecil, yaitu lebar 65 dan  
17     tinggi 92 piksel.
- 18     • **finga.png** gambar jari yang ditampilkan dihalaman awal.
- 19     • **GrapeYoda.png** gambar karakter anggur dengan ukuran besar, yaitu lebar 1746  
20     dan tinggi 2454 piksel.
- 21     • **grapeYodaTiny.png** gambar karakter anggur dengan ukuran kecil, yaitu lebar 65  
22     dan tinggi 92 piksel.
- 23     • **stepUp.png** gambar telapak kaki kiri yang ditampilkan dihalaman pengendali.
- 24     • **stepUpRight.png** gambar telapak kaki kanan yang ditampilkan dihalaman pe-  
25     ngendali.
- 26     • **grapeYodaTiny.png** gambar karakter anggur dengan ukuran kecil.
- 27     • **SummerEgg.png** gambar karakter telur mata sapi dengan ukuran besar, yaitu  
28     lebar 1746 dan tinggi 2454 piksel.
- 29     • **summerEggTiny.png** gambar karakter telur mata sapi dengan ukuran kecil, yaitu  
30     lebar 65 dan tinggi 92 piksel.
- 31     • **timer1.png** gambar angka 1 yang ditampilkan saat hitungan mundur.
- 32     • **timer2.png** gambar angka 2 yang ditampilkan saat hitungan mundur.
- 33     • **timer3.png** gambar angka 3 yang ditampilkan saat hitungan mundur.
- 34     • **track.png** gambar lintasan saat permainan dimulai.
- 35     • **winning.png** gambar panggung saat sudah ada pemenang dalam permainan.

1       (b) **Direktori javascripts**

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

5       i. **Direktori libs**

6       Direktori ini berisi berkas yang berfungsi untuk mengatur pengiriman data melewati  
7       form. Berikut isi dari folder ini:

8       • **jquery-3.3.1.min.js**

9       Berkas ini merupakan jQuery yang menyediakan fungsi-fungsi untuk pengiriman  
10      dan penerimaan data yang dilakukan *client* dan *server*.

11      ii. **charDesktopScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku  
12      halaman pemilihan karakter pada PC pada saat diakses oleh *client*.

13      Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- 14       • **marker**, *integer* yang menandakan ada berapa pemain yang sudah menetapkan  
15       karakter.
- 16       • **playerData**, *array* yang menyimpan objek pengguna dengan nilai karakter yang  
17       dipilihnya.
- 18       • **imageArray**, *array* yang menyimpan daftar karakter yang dapat dipilih.

19      Method yang dimiliki oleh berkas ini adalah sebagai berikut:

20       • **toGamePlayDesktop()**

21       Berfungsi untuk memindahkan halaman kehalaman memulai permainan.

22       • **beginWaiting(beginCounter)**

23       Berfungsi untuk melakukan hitungan mundur selama satu detik sebelum me-  
24       manggil method **toGamePlayDesktop()** untuk berpindah kehalaman memulai  
25       permainan.

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

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

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

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

31       Berfungsi untuk menangkap event *charSent* yang dipancarkan oleh *server* untuk  
32       memeriksa apakah kedua pemain telah menetapkan karakter untuk pindah  
33       kehalamannya selanjutnya.

34      iii. **charMobileScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku  
35      halaman pemilihan karakter pada smartphone pada saat diakses oleh *client*.

36      Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

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

39      Method yang dimiliki oleh berkas ini adalah sebagai berikut:

40       • **selectChar()**

41       Berfungsi untuk mengirimkan nilai karakter yang telah dipilih oleh pemain.

1     • **waitForCall(beginCounter)**

2       Berfungsi untuk melakukan hitungan mundur selama satu detik sebelum berpindah  
3       kehalamam memulai permainan.

4       Event yang dimiliki oleh berkas ini adalah sebagai berikut:

5     • **socket.on('charSent', function(msg))**

6       Berfungsi untuk menangkap event charSent yang dipancarkan oleh *server* untuk  
7       memeriksa apakah kedua pemain telah menetapkan karakter untuk pindah  
8       kehalamam selanjutnya.

9     • **socket.emit('selectingChar', val, id)**

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

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

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

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

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

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

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

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

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

22       Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

23     • **canvas**, elemen *canvas* yang ada pada berkas HTML.

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

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

28     • **track**, gambar lintasan lari.

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

30     • **timer2**, gambar angka 2 yang akan digunakan pada *countdown*.

31     • **timer3**, gambar angka 3 yang akan digunakan pada *countdown*.

32     • **player1Char**, gambar karakter milik pemain pertama.

33     • **player2Char**, gambar karakter milik pemain kedua.

34     • **dataOfPlayer**, menyimpan data kedua pemain.

35     • **player1Val**, menyimpan nilai karakter pemain pertama yang telah dipilih.

36     • **player2Val**, menyimpan nilai karakter pemain kedua yang telah dipilih.

37     • **winner**, menyimpan pemain pemenang.

38     • **timerArray**, menyimpan kumpulan gambar angka untuk proses *countdown*.

39     • **charArray**, menyimpan kumpulan gambar karakter.

40     • **arrOfPlayerChar**, menyimpan kedua karakter yang telah dipilih oleh pemain.

- 1     ● **aniFrame**, menyimpan *animation frame* yang digunakan untuk proses animasi karakter pemain pertama.
- 2     ● **aniFrame2**, menyimpan *animation frame* yang digunakan untuk proses animasi karakter pemain kedua.
- 3     ● **progressPlayer1**, menyimpan kemajuan suatu posisi karakter yang digerakan pada lintasan lari milik pemain pertama.
- 4     ● **progressPlayer2**, menyimpan kemajuan suatu posisi karakter yang digerakan pada lintasan lari milik pemain kedua.
- 5     ● **xPosition1**, menyimpan posisi karakter pemain pertama pada sumbu x.
- 6     ● **yPosition1**, menyimpan posisi karakter pemain pertama pada sumbu y.
- 7     ● **xPosition2**, menyimpan posisi karakter pemain kedua pada sumbu x.
- 8     ● **yPosition2**, menyimpan posisi karakter pemain kedua pada sumbu y.

13     Method yang dimiliki oleh berkas ini adalah sebagai berikut:

- 14     ● **init()**  
15         Berfungsi untuk menggambar lintasan lari diawal permainan.
- 16     ● **beginCountDown(beginCounter, msg)**  
17         Berfungsi untuk memulai hitungan mundur dengan menggambar angka tiga hingga satu kelayar permainan.
- 18     ● **drawChar(data)**  
19         Berfungsi untuk menggambar karakter milik pemain diatas lintasan lari.
- 20     ● **beginGamePlay()**  
21         Berfungsi untuk memulai permainan.
- 22     ● **reachFinishLine(player)**  
23         Berfungsi untuk mencatat pemain mana yang sampai pada garis akhir duluan.
- 24     ● **toWinningPage()**  
25         Berfungsi untuk memindahkan halaman ke halaman pemenang.
- 26     ● **readyPlayerOne()**  
27         Berfungsi untuk menggerakan karakter pemain pertama pada saat tombol telapak kaki ditekan.
- 28     ● **readyPlayerTwo()**  
29         Berfungsi untuk menggerakan karakter pemain kedua pada saat tombol telapak kaki ditekan.

33     Event yang dimiliki oleh berkas ini adalah sebagai berikut:

- 34     ● **socket.on('moveThePlayer', function(msg)...)**  
35         Berfungsi untuk menangkap *event moveThePlayer* yang dipancarkan oleh *server* saat tombol telapak kaki ditekan oleh pemain.
- 36     ● **socket.on('pong'), function(startTime){}**  
37         Berfungsi untuk menangkap *event pong* yang dipancarkan oleh *server* untuk menghitung *latency* pada saat karakter bergerak pada layar *PC*.
- 38     ● **socket.on('startTheGame', function(msg)...)**  
39         Berfungsi untuk menangkap *event startTheGame* yang dipancarkan oleh *se-*

2            *rver* pada saat akan memulai permainan dengan menggambar asset-asset yang  
3            dibutuhkan ke layar *PC*.

4            • **socket.on('toWinnerPage', function())**

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

7            • **socket.emit('sendingTheWinner', playerWin,playerCharArr)**

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

10          – **playerWin**, pemenang yang mencapai garis akhir lebih dulu.

11          – **playerCharArr**, nilai dari kedua karakter milik para pemain.

- 12         v. **gamePlayMobileScript.js**, Berkas ini berfungsi untuk mengatur bagaimana peri-  
13            laku halaman permainan di *mobile browser* pada saat diakses oleh *client*.

14            Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

15          • **stepLeftHtml**, Elemen HTML berupa gambar telapak kaki.

16          • **instructionEl**, Elemen HTML berupa teks instruksi.

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

18          • **stepClicked()**

19            Berfungsi untuk menangani aksi menekan tombol telapak kaki oleh pemain.

20          • **moveToWinPage()**

21            Berfungsi untuk memindahkan halaman ke halaman pemenang.

22          • **showInstruction(beginCounter)**

23            Berfungsi untuk menampilkan instruksi kelayar permainan.

24            **Parameter:**

25          – **beginCounter**, *integer* yang menunjukkan berapa detik instruksi akan ditam-  
26            pilkan.

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

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

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

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

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

34          • **socket.emit('ping'), startTime**

35            Berfungsi untuk mengirimkan *event* **ping** pada saat tombol telapak kaki ditekan,  
36            untuk menghitung *latency*. Parameter *startTime* adalah waktu saat ini pada  
37            saat tombol ditekan.

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

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

- 41         vi. **homeScript.js**, Berkas ini berfungsi untuk mengatur bagaimana perilaku halaman  
1            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 menunjukkan pemain urutan keberapa.

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

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

2           Method yang dimiliki oleh berkas ini adalah sebagai berikut:

3           • **toCharDesk()**

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

6           • **getRandInt()**

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

8           • **requestAccepted(msg)**

9           Berfungsi untuk memasukan data *client* kedalam *array* pada saat telah tersam-  
10          bung ke Socket.io.

11         Event yang dimiliki oleh berkas ini adalah sebagai berikut:

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

13         Berfungsi untuk menangkap *event connect* pada saat *client* telah berhasil ter-  
14          sambung ke Socket.io.

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

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

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

19         Berfungsi untuk menangkap *event toCharPage* yang dipancarkan oleh *server*  
20          pada saat akan memindahkan halaman ke halaman pemilihan karakter pada  
21          *smartphone*

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

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

25           – **id** identifikasi unik Socket.io milik *client*

26           – **room** kode *room* milik *client*

27         • **socket.emit('roomFull', room)**

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

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

33         Atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

34           • **winCanvas** objek untuk mengambil elemen dengan *id* canvasWinStage.

35           • **winCtx** objek 2d *canvas*.

36           • **podium** gambar podium yang akan diletakan pada *canvas*.

37           • **winnerCharArr** *array* yang akan menampung data milik para pemain.

38         Method yang dimiliki oleh berkas ini adalah sebagai berikut:

39           • **drawStage()**

40           Berfungsi untuk menggambar podium ke layar permainan.

41           • **drawWinner(winner)**

1           Berfungsi untuk menggambar para pemenang permainan.

2     • **drawFirstWinner(firstWinner)**

3       Berfungsi untuk menggambar pemenang pertama.

4     • **drawSecondWinner(secondWinner)**

5       Berfungsi untuk menggambar pemenang kedua.

6     • **toHomePage()**

7       Berfungsi untuk memindahkan halaman ke halaman awal.

8       Event yang dimiliki oleh berkas ini adalah sebagai berikut:

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

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

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

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

13     • **socket.emit('goBackHome', 'back home')**

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

- 15   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:

16     • **game\_over**, elemen HTML berupa teks yang memiliki *id gameOver*.

17       Method yang dimiliki oleh berkas ini adalah sebagai berikut:

18     • **backToHome()**

19       Berfungsi untuk memindahkan halaman ke halaman awal.

20       Event yang dimiliki oleh berkas ini adalah sebagai berikut:

21     • **socket.on('getTheWinner',function(msg))**, berfungsi untuk menangkap event getTheWinner yang kemudian akan mengeksekusi fungsi untuk menangani event tersebut.

22     • **socket.on('backHome', function(msg))**, berfungsi untuk menangkap event backHome yang akan mengeksekusi fungsi untuk kembali kehalaman utama.

23     • **socket.on('winning', function(msg))**, berfungsi untuk menangkap event winning dan akan menampilkan pesan '*YOU WIN*'.

24     • **socket.on('losing', function(msg))**, berfungsi untuk menangkap event losing dan akan menampilkan pesan '*YOU LOSE*'.

25   (c) **Folder stylesheets**

26       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:

- 27     i. **charDesktopStyle.css**, Berkas ini berfungsi untuk menghias halaman pemilihan karakter pada *PC browser*.

- 28     ii. **charMobileStyle.css**, Berkas ini berfungsi untuk menghias halaman pemilihan karakter pada *mobile browser*.

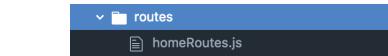
- 2       iii. **gamePlayDesktopStyle.css**, Berkas ini berfungsi untuk menghias halaman permainan pada *PC browser*.
- 3       iv. **gamePlayMobileStyle.css**, Berkas ini berfungsi untuk menghias halaman permainan pada *mobile browser*.
- 4       v. **homeStyle.css**, Berkas ini berfungsi untuk menghias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

### 18       3. Folder routes

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

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

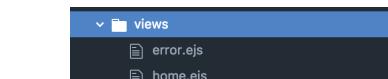
### 25       4. Folder views



Gambar 4.18: Isi dari folder routes

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

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



Gambar 4.19: Isi dari folder views

27       Folder ini berisi bagian-bagian yang berfungsi untuk menampilkan halaman-halaman kepada  
1       *client*. Folder ini memiliki berkas dengan ekstensi **.ejs**. Sama seperti ekstensi **.html**, ekstensi  
2       tersebut akan menampilkan berbagai tampilan dengan menggunakan *syntax .html*. Berikut  
3       merupakan isi dari folder ini:

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

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

8       **5. Berkas app.js**

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

11      Beberapa atribut yang dimiliki oleh berkas ini adalah sebagai berikut:

- 12      • **express**, atribut ini akan mengakses seluruh fitur yang disediakan oleh modul '*express*'.  
13      • **path**, atribut ini akan mengakses seluruh fitur yang disediakan oleh modul '*path*'.  
14      • **cookieParser**, atribut ini akan mengakses seluruh fitur yang disediakan oleh modul  
15       '*cookie-parser*'.  
16      • **homeRouter**, atribut ini akan mengakses seluruh fitur yang disediakan oleh modul  
17       '*homeRoutes*'.

18      Method yang dimiliki oleh berkas ini adalah sebagai berikut:

- 19      • **app.set('views', path.join(\_\_dirname, 'views'))**, method ini akan mengatur pe-  
1       nampilan halaman web agar menggunakan seluruh berkas yang berada didalam folder  
2       *views*.  
3      • **app.set('view engine', 'ejs')**, method ini akan menetapkan *view engine* dengan nilai  
4       *ejs*.  
5      • **app.use('/', homeRouter)**, method ini akan menetapkan akses yang diberikan kepada  
6       *client* apabila mengakses alamat web Finger For Life. Kembalian akan berupa variabel  
7       *homeRouter* yang akan menampilkan halaman utama dari web.

8

## BAB 5

9

### IMPLEMENTASI DAN PENGUJIAN

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

14 **5.1 Implementasi**

15 **5.1.1 Lingkungan Implementasi**

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

18 **1. Lingkungan Perangkat Keras**

19

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

24 **2. Lingkungan Perangkat Lunak**

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

27 (a) **Server**

1

2 **i. Heroku**

3 Heroku adalah *cloud platform* yang menyediakan fitur-fitur yang membantu pengguna  
4 untuk dapat memiliki alamat domain <sup>1</sup>. Dengan menggunakan Heroku, pengembang  
5 aplikasi dapat menaruh aplikasi yang sudah dibuat seluruhnya di *cloud*. Heroku  
6 dapat menjalankan aplikasi berbasis beberapa bahasa pemrograman. Salah satu  
7 bahasa pemrograman yang didukung oleh Heroku adalah Node.js.

---

<sup>1</sup><https://www.heroku.com/what>

Dengan menggunakan Heroku, proses konfigurasi, penyesuaian, dan pengaturan aplikasi dapat dilakukan secara sederhana dan efisien. Proses infrastruktur dalam pengaturan *URL* akan sepenuhnya dilakukan oleh Heroku. Dengan begitu, pengembang aplikasi dapat fokus terhadap pembangunan aplikasi.

Spesifikasi Heroku yang digunakan didalam pengembangan Finger For Life adalah 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>.

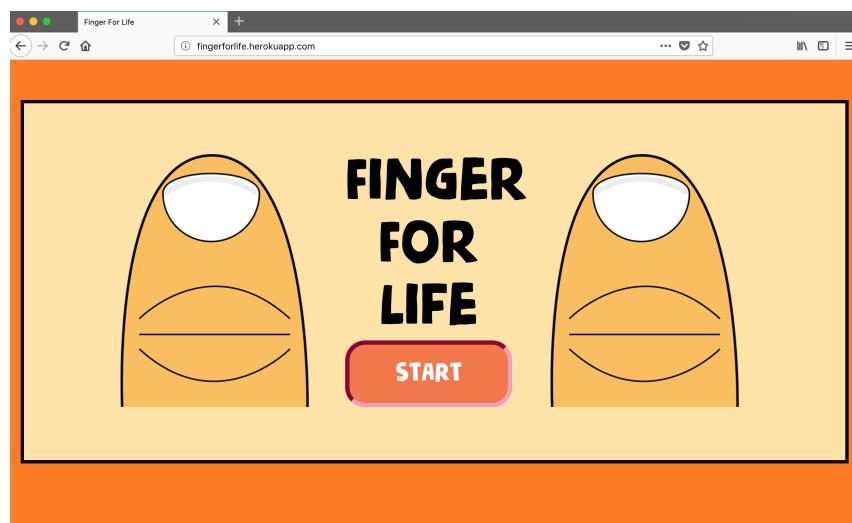
### 5.1.2 Hasil Implementasi

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 Utama

##### PC

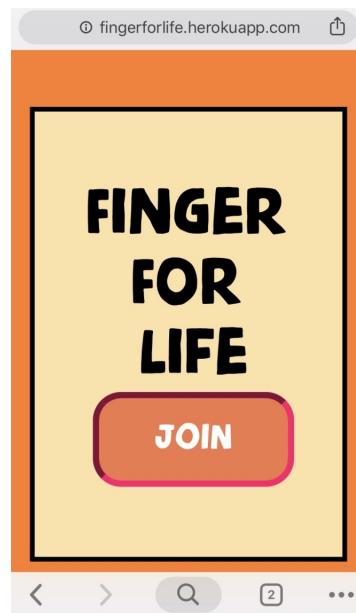
Halaman utama pada *PC* digunakan pengguna untuk memulai permainan. Pada halaman ini terdapat tombol *start* yang berfungsi untuk meminta akses masuk kedalam permainan. Apabila pengguna menekan tombol tersebut, maka pengguna akan diarahkan kehalaman selanjutnya. Tangkapan layar dari halaman utama pada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



Gambar 5.1: Halaman utama pada *PC*

##### Smartphone

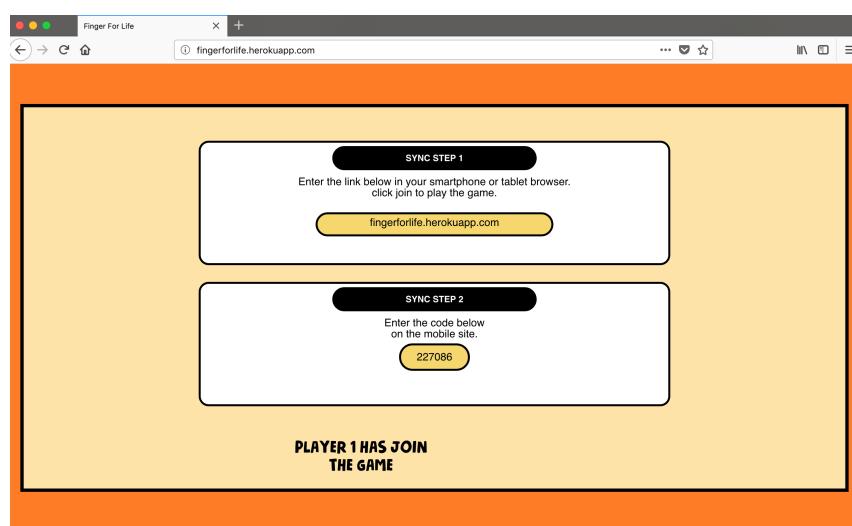
Halaman utama pada *smartphone* digunakan pengguna untuk memulai permainan. Pada halaman ini terdapat tombol *join* yang berfungsi untuk meminta akses masuk kedalam permainan. Apabila pengguna menekan tombol tersebut, maka pengguna akan diarahkan kehalaman selanjutnya. Tangkapan layar dari halaman utama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



Gambar 5.2: Halaman utama pada *smartphone*

8     2. Halaman permintaan bergabung  
9        PC

10      Halaman permintaan bergabung pada *PC* berguna untuk proses sinkronisasi antara *PC* dan  
11      *smartphone*. Halaman ini menyediakan kode *room* yang harus dikirimkan oleh pengguna  
1      melalui *smartphone*. Apabila kode yang dikirimkan pengguna sesuai, maka pengguna akan  
2      bergabung kedalam *room* permainan. Apabila kode tidak sesuai, maka pengguna tidak bisa  
3      bergabung kedalam *room* permainan. Tangkapan layar dari halaman permintaan bergabung  
4      pada *PC* dapat dilihat pada gambar 5.3.

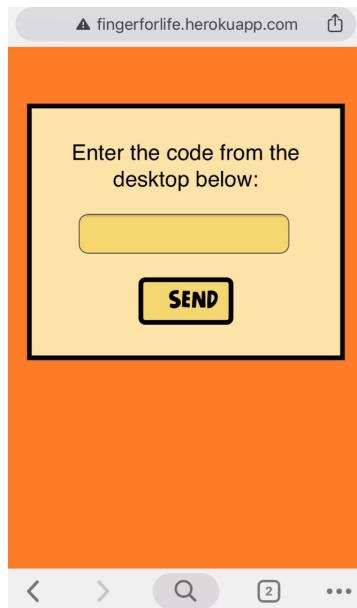


Gambar 5.3: Halaman permintaan bergabung pada *PC*

5        **Smartphone**

6      Halaman permintaan bergabung pada *smartphone* berguna untuk proses sinkronisasi antara  
7      *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 menunjukan berhasil bergabung kedalam *room*. Apabila tidak sesuai, maka akan muncul teks yang menunjukan tidak berhasil bergabung kedalam *room*. Tangkapan layar dari halaman permintaan bergabung pada *PC* dapat dilihat pada gambar 5.4.

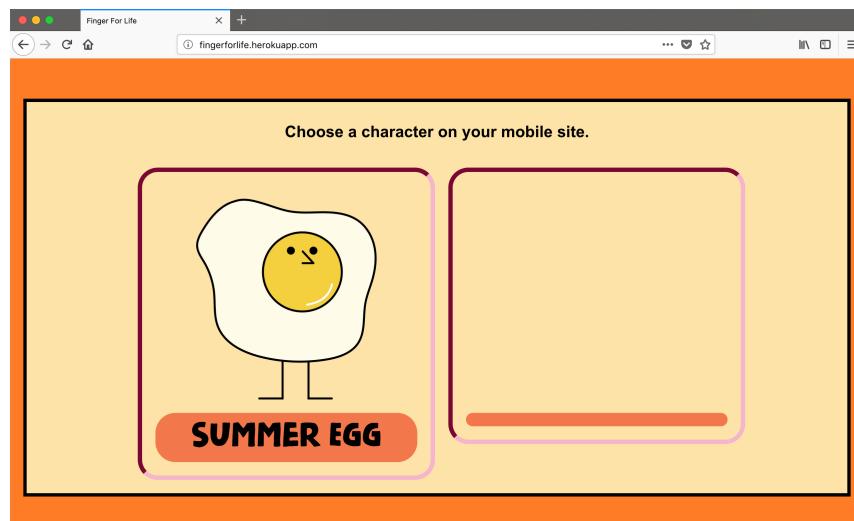


Gambar 5.4: Halaman permintaan bergabung pada *smartphone*

### 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.



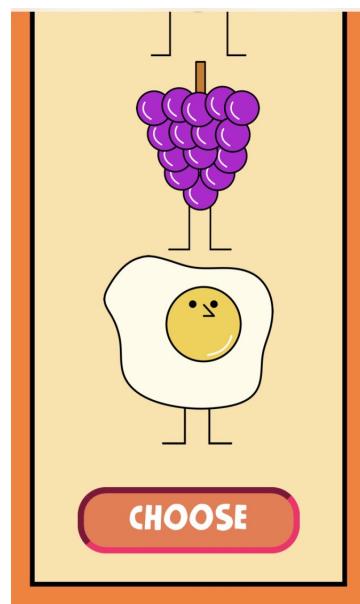
Gambar 5.5: Halaman memilih karakter pada *PC*

### 8      **Smartphone**

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



(a) Halaman memilih karakter 1



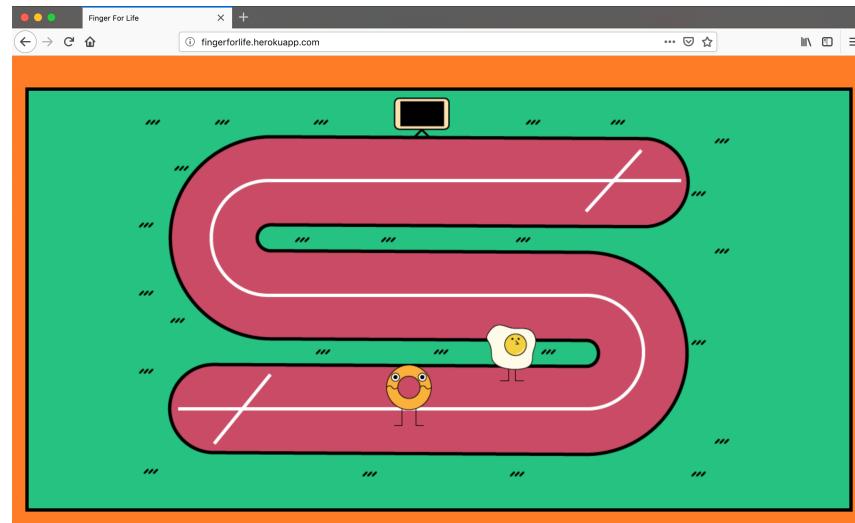
(b) Halaman memilih karakter 2

Gambar 5.6: Halaman memilih karakter pada *smartphone*

### 2      4. Halaman Memulai Permainan

#### 3      **PC**

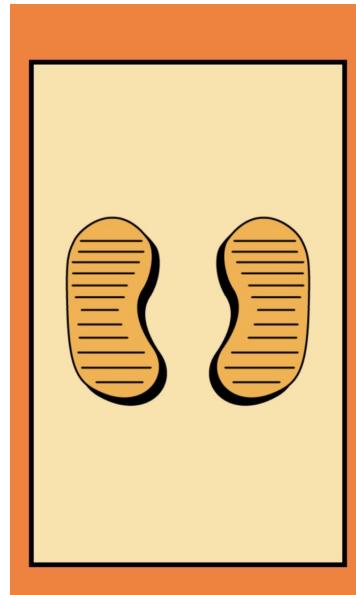
4      Halaman ini menampilkan lintasan lari dan kedua karakter yang dapat dimainkan. Karakter  
5      pada layar dapat bergerak melalui lintasan lari apabila pemain menekan tombol yang ada  
6      pada halaman *smartphone*. Tangkapan layar dari halaman memulai permainan pada *PC*  
7      dapat dilihat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



Gambar 5.7: Halaman memulai permainan pada *PC*

### 8      **Smartphone**

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



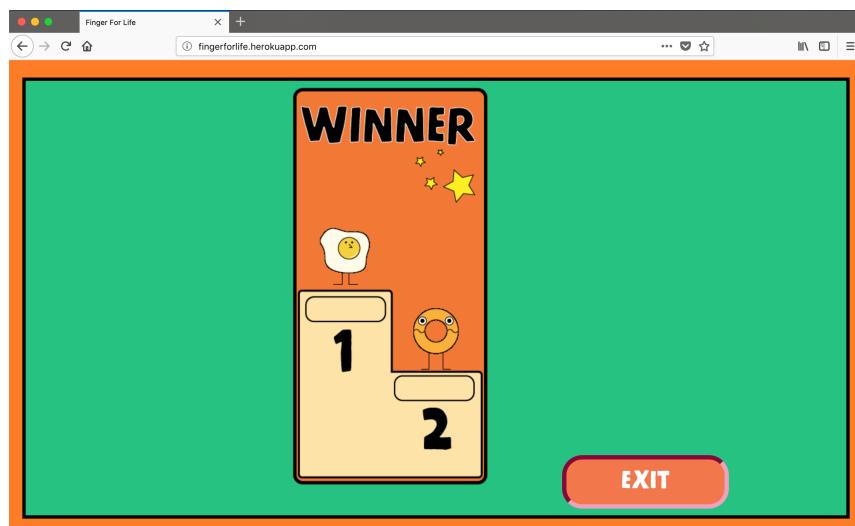
Gambar 5.8: Halaman memulai permainan pada *smartphone*

### 3      **5. Halaman Mengakhiri Permainan**

#### 4      **PC**

5      Halaman ini menampilkan pemain yang memenangkan permainan. Pemenang pertama akan  
6      ditampilkan dipodium posisi atas, sedangkan pemenang kedua akan ditampilkan dipodium  
1      posisi bawah. Pemain dapat menekan tombol *exit* untuk keluar dari permainan. Tangkapan

- 2 layar dari halaman mengakhiri permainan pada *PC* dapat dilihat pada gambar 5.9.



Gambar 5.9: Halaman mengakhiri permainan pada *PC*

3 **Smartphone**

4 Halaman ini menampilkan teks yang menunjukan posisi pemenang. Pemenang pertama  
5 akan ditampilkan teks yang menunjukan posisi pertama, sedangkan pemenang kedua akan  
6 ditampilkan teks yang menunjukan posisi kedua. Tangkapan layar dari halaman mengakhiri  
7 permainan pada *smartphone* dapat dilihat pada gambar 5.10.



Gambar 5.10: Halaman mengakhiri permainan pada *smartphone*

## 8    5.2 Pengujian

### 9    5.2.1 Pengujian Fungsional

10 Pengujian fungsional dilakukan untuk mengetahui kesesuaian reaksi perangkat lunak dengan reaksi  
11 yang diharapkan berdasarkan aksi pengguna terhadap perangkat lunak. Pada saat pengujian  
12 fungsional dilakukan, penulis menemukan beberapa masalah yang muncul pada program aplikasi.  
13 Masalah tersebut muncul pada beberapa tahapan saat program dijalankan. Berdasarkan masalah-  
14 masalah yang ditemukan, penulis juga mendapatkan solusi atas masalah yang ada. Berikut akan  
15 dijelaskan beberapa masalah yang ditemukan, beserta proses memecahkan masalah yang telah  
16 dilakukan.

17    1. Halaman utama

- 18    • **Masalah:**

19    Tampilan halaman utama pada *smartphone* terkadang muncul gambar jari, dimana  
20    seharusnya gambar jadi hanya muncul dihalaman utama pada *PC*. Masalah ini ditemukan  
21    pada *smartphone* dengan resolusi layar yang besar, yaitu dengan ukuran lebar lebih dari  
22    1260 piksel.

- 23    • **Solusi:**

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

```
27 @media (max-width:1260px){  
28     #left, #right, #startButton{  
29         display: none;  
30         float: none;  
31     }  
32  
33     #joinButton{  
34         visibility: visible;  
35         width: 85%;  
36     }  
1 }
```

Listing 5.1: Fitur CSS *@media querry*

2    2. Halaman permintaan bergabung

- 3    • **Masalah:**

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

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

- 14      • **Solusi:**

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

```
18     function requestToJoin(){
19         socket .emit( 'requestToJoin' , {
20             id: socket .id ,
21             room: $( '#code' ) .val ()
22         });
23     }
```

Listing 5.2: Proses memancarkan *event*

### 24      3. Halaman pemilihan karakter

- 25      • **Masalah:**

26       Pada saat salah satu pemain menekan tombol `choose` dua kali, halaman langsung pindah  
 27      kehalaman memulai permainan disaat pemain kedua belum memilih karakter.

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

- 33      • **Solusi:**

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

```
39     var valButton = $( 'input [name="radioChar"] :checked' ) .val ();
40       if ( valButton === undefined ) {
41           alert ( "You have not choose a character." );
42       }
43       else {
44           socket .emit( 'sendingChar' , {
45             val: valButton ,
46             id: socket .id ,
```

```

8     marker: 1
9   });
10  document.getElementById("nextButton").disabled = true;
11}

```

Listing 5.3: Proses menangani tombol *choose*

12 **4. Halaman pemilihan karakter**

13 • **Masalah:**

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

16  
17 *Error* ini terjadi karena adanya kesalahan pengiriman data pada saat proses bergabung  
18 kedalam *room*. Yang masuk kedalam *room* hanya satu pemain saja, sedangkan pemain  
19 kedua tidak mengirimkan data kepada *server* sehingga tidak masuk kedalam *room*.

20 • **Solusi:**

21 Solusi dari masalah ini adalah dengan menggunakan JavaScript, untuk menangani  
22 proses permintaan bergabung agar kedua pemain dapat masuk kedalam room. Berikut  
23 merupakan potongan kode dari solusi masalah ini:

```

24 function requestToJoin(){
25   socket.emit('requestToJoin', {
26     id: socket.id,
27     room: $('#code').val()
28   });
29 }

```

Listing 5.4: Proses menangani memancarkan *event*

2 Pengujian fungsional yang dilakukan terbagi menjadi dua, yaitu pengujian fungsional pada *PC*  
3 dan pada *smartphone*.

4 **PC**

5

Tabel 5.1: Tabel Pengujian Fungsional pada *PC*

No.	Aksi Pengguna	Reaksi yang diharapkan	Reaksi perangkat lunak
1	Pengguna menjalankan aplikasi	Halaman utama 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 kedalam <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

6

**Smartphone**

7

Tabel 5.2: Tabel Pengujian Fungsional pada *smartphone*

No.	Aksi Pengguna	Reaksi yang diharapkan	Reaksi perangkat lunak
1	Pengguna menjalankan aplikasi	Halaman utama 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 utama	sesuai

### 5.2.2 Pengujian Eksperimental

Pengujian eksperimental dilakukan terhadap beberapa responden dengan *smartphone* dan *PC* yang berbeda-beda. Setiap responden diminta untuk mengakses URL yang diberikan melalui *smartphone* dan *PC* milik masing-masing dan memastikan apakah fungsi dari setiap halaman yang ditampilkan sudah berjalan. Pengujian eksperimental terbagi menjadi dua, yaitu pengujian *cross-platform* dan pengujian *latency*.

#### 1. Pengujian *Cross-Platform*

Pengujian *cross-platform* merupakan pengujian yang dilakukan terhadap jenis *smartphone* dan jenis *PC* yang berbeda-beda. Pengujian ini dilakukan untuk memastikan setiap fungsi dari setiap halaman yang ditampilkan sudah berjalan diseluruh *platform*. Dari beberapa responden yang sudah melakukan pengujian, didapatkan hasil sebagai berikut:

##### (a) Pengujian pertama

Jenis *smartphone* pemain 1: iPhone 7

Jenis *browser* pemain 1: Safari

Jaringan internet pemain 1: Telkomsel

8           Jenis *smartphone* pemain 2: iPhone 7 Plus

9           Jenis *browser* pemain 2: Google Chrome

10          Jaringan internet pemain 2: Telkomsel

11          Jenis *PC*: Macbook 13" mid 2014

12          Jenis *browser PC*: Google Chrome

13          Jaringan internet *PC*: Wifi UNPAR9

14          Hasil: Berdasarkan tabel 5.1 dan 5.2 yang ada pada subbab 5.2.1, seluruh fungsi telah  
15          berjalan dengan baik.

16          (b) **Pengujian kedua**

17          Jenis *smartphone* pemain 1: Samsung Galaxy S4

18          Jenis *browser* pemain 1: Google Chrome

19          Jaringan internet pemain 1: Wifi Megavision

20          Jenis *smartphone* pemain 2: Motorola Moto G (5S Plus)

21          Jenis *browser* pemain 2: Dolphin Browser

22          Jaringan internet pemain 2: Wifi Megavision

23          Jenis *PC*: Asus A450L

24          Jenis *browser PC*: Google Chrome

25          Jaringan internet *PC*: Wifi Megavision

26          Hasil: Berdasarkan tabel 5.1 dan 5.2 yang ada pada subbab 5.2.1, seluruh fungsi telah  
27          berjalan dengan baik.

28          (c) **Pengujian ketiga**

29          Jenis *smartphone* pemain 1: iPhone 7 Plus

30          Jenis *browser* pemain 1: Google Chrome

31          Jaringan internet pemain 1: Telkomsel

32          Jenis *smartphone* pemain 2: iPhone 7 Plus

33          Jenis *browser* pemain 2: Safari

34          Jaringan internet pemain 2: Telkomsel

35          Jenis *PC*: Asus ROG FX

36          Jenis *browser PC*: Google Chrome

37          Jaringan internet *PC*: Wifi UNPAR9

38          Hasil: Berdasarkan tabel 5.1 dan 5.2 yang ada pada subbab 5.2.1, seluruh fungsi telah  
39          berjalan dengan baik.

40          (d) **Pengujian keempat**

41          Jenis *smartphone* pemain 1: iPhone 7

8           Jenis *browser* pemain 1: Google Chrome  
9           Jaringan internet pemain 1: Telkomsel  
10  
11          Jenis *smartphone* pemain 2: Xiaomi mi note 3  
12          Jenis *browser* pemain 2: Google Chrome  
13          Jaringan internet pemain 2: Telkomsel  
14  
15          Jenis *PC*: Macbook Pro 13" Mid 2014  
16          Jenis *browser PC*: Google Chrome  
17          Jaringan internet *PC*: Wifi The Kiosk  
18  
19          Hasil: Berdasarkan tabel 5.1 dan 5.2 yang ada pada subbab 5.2.1, seluruh fungsi telah  
20         berjalan dengan baik.

21         (e) **Pengujian kelima**

22          Jenis *smartphone* pemain 1: Samsung J7 Pro  
23          Jenis *browser* pemain 1: Google Chrome  
24          Jaringan internet pemain 1: Telkomsel  
25  
26          Jenis *smartphone* pemain 2: iPhone 7 Plus  
27          Jenis *browser* pemain 2: Google Chrome  
28          Jaringan internet pemain 2: Telkomsel  
29  
30          Jenis *PC*: Asus ROG FX  
31          Jenis *browser PC*: Google Chrome  
32          Jaringan internet *PC*: Wifi UNPAR9

33  
34          Hasil: Berdasarkan tabel 5.1 dan 5.2 yang ada pada subbab 5.2.1, seluruh fungsi telah  
35         berjalan dengan baik.

36          Dari beberapa pengujian yang dilakukan, dapat disimpulkan bahwa aplikasi telah berjalan  
37         dengan baik di beberapa *platform* yang berbeda.

38         2. **Pengujian *latency***

39          *Latency* merupakan kecepatan yang dihasilkan pada saat pengiriman data dari *client* menuju  
1         *server* maupun sebaliknya. Pengujian ini dilakukan untuk meneliti jumlah *latency* yang  
2         dihasilkan dari implementasi Socket.io. Pengujian *latency* yang dilakukan berfokus pada  
3         halaman memulai permainan, saat kedua pemain memainkan permainan Finger For Life.  
4         *Latency* yang dihitung adalah perbandingan waktu antara pemain menekan tombol kaki pada  
5         layar *smartphone* hingga karakter pada layar *PC* bergerak.

6          Dari beberapa responden yang melakukan pengujian, didapatkan beberapa hasil seperti  
7         berikut:

8       (a) **Pengujian Pertama**

9       Jenis *smartphone* pemain 1: Samsung J7

10      Jenis *browser* pemain 1: Internet (Default Browser Android)

11      Jaringan internet pemain 1: Wifi UNPAR2

13      Jenis *smartphone* pemain 2: LG Q6+

14      Jenis *browser* pemain 2: Google Chrome

15      Jaringan internet pemain 2: Wifi UNPAR2

17      Jenis *PC*: HP Pavilion G4

18      Jenis *browser PC* : Mozilla Firefox

19      Jaringan internet *PC*: Wifi UNPAR2

21      Hasil: Berdasarkan percobaan yang telah dilakukan, jumlah *latency* yang dihasilkan oleh  
22      pemain 1 adalah 203 milidetik, dan jumlah *latency* yang dihasilkan oleh pemain 2 adalah  
23      203 milidetik.

24       (b) **Pengujian Kedua**

25      Jenis *smartphone* pemain 1: iPhone 7

26      Jenis *browser* pemain 1: Google Chrome

27      Jaringan internet pemain 1: Telkomsel

29      Jenis *smartphone* pemain 2: Xiaomi Mi Note 3

30      Jenis *browser* pemain 2: Google Chrome

31      Jaringan internet pemain 2: Telkomsel

33      Jenis *PC*: Macbook Pro 13" Mid 2014

34      Jenis *browser PC* : Google Chrome

35      Jaringan internet *PC*: Wifi Telkomsel

37      Hasil: Berdasarkan percobaan yang telah dilakukan, jumlah *latency* yang dihasilkan oleh  
38      pemain 1 adalah 277 milidetik, dan jumlah *latency* yang dihasilkan oleh pemain 2 adalah  
39      298 milidetik.

40       (c) **Pengujian Ketiga**

41      Jenis *smartphone* pemain 1: coolpad e502

1      Jenis *browser* pemain 1: Google Chrome

2      Jaringan internet pemain 1: Telkomsel

4      Jenis *smartphone* pemain 2: Oppo F7

5      Jenis *browser* pemain 2: Google Chrome

6      Jaringan internet pemain 2: Telkomsel

8           Jenis *PC*: Macbook Pro 13" Mid 2014

9           Jenis *browser PC* : Google Chrome

10          Jaringan internet *PC*: Wifi Telkomsel

11  
12          Hasil: Berdasarkan percobaan yang telah dilakukan, jumlah *latency* yang dihasilkan oleh  
13          pemain 1 adalah 556 milidetik, dan jumlah *latency* yang dihasilkan oleh pemain 2 adalah  
14          567 milidetik.

15          (d) **Pengujian Keempat**

16          Jenis *smartphone* pemain 1: Sony Xperia M5

17          Jenis *browser* pemain 1: Google Chrome

18          Jaringan internet pemain 1: Wifi UNPAR2

20          Jenis *smartphone* pemain 2: Oppo F3

21          Jenis *browser* pemain 2: Google Chrome

22          Jaringan internet pemain 2: Wifi UNPAR2

23          Jenis *PC*: Macbook Pro 13" Mid 2014

1          Jenis *browser PC* : Google Chrome

2          Jaringan internet *PC*: Wifi UNPAR2

4  
5          Hasil: Berdasarkan percobaan yang telah dilakukan, jumlah *latency* yang dihasilkan oleh  
6          pemain 1 adalah 398 milidetik, dan jumlah *latency* yang dihasilkan oleh pemain 2 adalah  
7          396 milidetik.



## BAB 6

### KESIMPULAN DAN SARAN

#### 10 6.1 Kesimpulan

- 11 Dari hasil pembangunan aplikasi Finger For Life, didapatkanlah kesimpulan-kesimpulan sebagai  
 12 berikut:
- 13 1. Aplikasi Finger For Life telah berhasil dibangun dengan menggunakan Socket.io. Selain  
 14 itu, aplikasi ini dibangun berdasarkan Node.js, dan juga menggunakan Express.js. Untuk  
 15 proses animasi, Finger For Life menggunakan CanvasAPI dalam proses pengembangannya.  
 16 Dalam proses pengolahan halaman HTML pada aplikasi, proses tersebut dibantu dengan  
 17 menggunakan jQuery dan The Content Template Element dari HTML.
  - 18 2. Telah berhasil mendapatkan jumlah *latency* yang dihasilkan oleh aplikasi Finger For Life  
 19 dengan pengimplementasian Socket.io. Rata-rata jumlah *latency* yang dihasilkan adalah 362  
 20 *milliseconds*, atau sekitar 0,362 detik.

#### 21 6.2 Saran

- 22 Dari hasil penelitian termasuk kesimpulan yang didapat, berikut adalah saran untuk pengembangan  
 23 lebih lanjut:
- 24 1. Penelitian ini menggunakan metode berbasis *event* yang diimplementasi menggunakan Socket.io  
 2624 untuk menghitung jumlah *latency* yang dihasilkan oleh aplikasi Finger For Life pada saat  
 2625 dimainkan. Proses menghitung *latency* digunakan dengan menggunakan waktu yang terdapat  
 2626 pada *PC* sebagai acuan waktunya. Dengan demikian, hasil *latency* yang didapat terkadang  
 2627 berjumlah negatif. Hal tersebut terjadi karena waktu yang terdapat pada *PC* dan *smartphone*  
 2628 tidak sinkron. Oleh karena itu, dibutuhkan metode yang lebih baik agar waktu yang terdapat  
 2629 pada *PC* dan *smartphone* akan selalu sinkron, sehingga tidak menghasilkan *latency* dengan  
 2630 bilangan negatif.



## **DAFTAR REFERENSI**

- [1] Arrachequesne, D. (2011) Socket.io Docs. <https://socket.io/docs/>. 3 September 2018.
- [2] Mozilla dan Contributors, I. (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] Foundation, N. (2009) Node.js Docs. <https://nodejs.org/en/docs/>. 3 September 2018.
- [4] Foundation, N. (2010) Express.js 4.x API. <https://expressjs.com/en/4x/api.html>. 3 September 2018.
- [5] Mozilla dan Contributors, I. (2005) The Content Template element. <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template>. 3 September 2018.
- [6] jQuery Foundation, T. (2006) jQuery API. <https://api.jquery.com/>. 3 September 2018.



# LAMPIRAN A

## KODE PROGRAM CLIENT

### A.1 Kode Program Direktori *public*

#### A.1.1 Kode Program Halaman Utama

##### A.1.1.1 Kode Program JavaScript Halaman Utama pada *PC*

Listing A.1: homeScript.js

```
1 var bg = $(".stage-area");
2
3 var titleHtml = $("#homePage").html();
4 bg.html(titleHtml);
5
6 function startClicked(){
7     var syncHtml = $("#syncPage").html();
8     bg.html(syncHtml);
9 }
10
11 function joinClicked(){
12     var mobileHtml = $("#mobilePage").html();
13     bg.html(mobileHtml);
14 }
```

##### A.1.1.2 Kode Program CSS Halaman Utama pada *PC*

Listing A.2: homeStyle.css

```
1 /* Globals */
2
3 body{
4     background-color: #FF7C27;
5     margin: 0;
6     padding: 0;
7 }
8
9 @font-face {
10     font-family: "fingerfunny";
11     src: url("Nickname_DEMO.otf");
12 }
13
14 .stage-area{
15     border-style: solid;
16     border-color: black;
17     border-width: thick;
18     margin: 20px;
19     margin-top: 60px;
20     padding: 10px;
21     background-color: #FDE3A7;
22     /* border: 5px solid blue; */
23 }
24 /* ----- */
25
26 /* Home Page */
27 .container{
28     width:90%;
29     margin:auto;
30     padding-top: 50px;
31     padding-bottom: 50px;
32     overflow:hidden;
33     /* border: 2px solid red; */
34 }
35
36 /* #titles{
37     margin-top:50px;
38     border: 2px solid black;
39 } */
40
41 #titles .title_container{
42     width: 90%;
43     margin: auto;
```

```

44    overflow: hidden;
45    text-align: center;
46    /* border: 2px solid yellow; */
47 }
48
49 #titles .title{
50    float:left;
51    text-align: center;
52    width:30%;
53    margin: 4px;
54    padding:10px;
55    /* border: 2px solid black; */
56 }
57
58 #titles .title .word{
59    float: left;
60    text-align: center;
61    display: block;
62    /* border: 2px solid green; */
63    width: 250px;
64    margin: 0;
65    margin-left: 20px;
66    padding: 0;
67 }
68
69 #titles .title .buttons{
70    float: left;
71    width: 250px;
72    margin-left: 20px;
73    /* border: 2px solid orange; */
74 }
75
76 #titles .title .word p{
77    font-family:'fingerfunny';
78    font-size: 70pt;
79    text-align: center;
80    padding: 0;
81    margin: 0;
82 }
83
84 #startButton, #joinButton{
85    width: 100%;
86    height: 100px;
87    padding: 0;
88    border-radius: 30px;
89    border-width: 5pt;
90    border-color: #DB0A5B;
91    border-style:inset;
92    background-color: #F2784B;
93    font-family: "fingerfunny";
94    text-align: center;
95    font-size: 30pt;
96    color: white;
97 }
98
99 #startButton:hover , #joinButton:hover{
100    background-color: #F54707;
101 }
102
103 #titles .title object{
104    height:380px;
105    /* border: 2px solid purple; */
106 }
107
108 @media (max-width:1260px){
109    #left, #right, #startButton{
110        display: none;
111        float: none;
112    }
113
114    #joinButton{
115        visibility: visible;
116        width: 85%;
117    }
118 }
119
120 @media (max-width:800px){
121    #left, #right{
122        display: none;
123    }
124
125    #titles .title .word, #titles .title .buttons{
126        margin-left: -10px;
127    }
128
129    #titles .title .word p{
130        font-size: 55pt;
131    }
132 }
133
134 @media (min-width:1260px) {
135    #joinButton{
136        display: none;
137    }
138 }
```

#### A.1.1.3 Kode Program JavaScript Halaman Utama pada *Smartphone*

Listing A.3: mobileScript.js

```

1 var bg = $(".stage-area");
2
3 var titleHtml = $("#homePage").html();
4 bg.html(titleHtml);
5
6 function startClicked(){
7     var syncHtml = $("#syncPage").html();
8     bg.html(syncHtml);
9 }
10
11 function joinClicked(){
12     var mobileHtml = $("#mobilePage").html();
13     bg.html(mobileHtml);
14 }
```

#### A.1.1.4 Kode Program CSS Halaman Utama pada *Smartphone*

Listing A.4: mobileStyle.css

```

1 /* Globals */
2
3 body{
4     background-color: #FF7C27;
5     margin: 0;
6     padding: 0;
7 }
8
9 @font-face {
10     font-family: "fingerfunny";
11     src: url("Nickname_DEMO.otf");
12 }
13
14 .stage-area{
15     border-style: solid;
16     border-color: black;
17     border-width: thick;
18     margin: 20px;
19     margin-top: 60px;
20     padding: 10px;
21     background-color: #FDE3A7;
22     /* border: 5px solid blue; */
23 }
24 /* ----- */
25
26 /* Home Page */
27 .container{
28     width:90%;
29     margin:auto;
30     padding-top: 50px;
31     padding-bottom: 50px;
32     overflow:hidden;
33     /* border: 2px solid red; */
34 }
35
36 /* #titles{
37     margin-top:50px;
38     border: 2px solid black;
39 } */
40
41 #titles .title_container{
42     width: 90%;
43     margin: auto;
44     overflow: hidden;
45     text-align: center;
46     /* border: 2px solid yellow; */
47 }
48
49 #titles .title{
50     float:left;
51     text-align: center;
52     width:30%;
53     margin: 4px;
54     padding:10px;
55     /* border: 2px solid black; */
56 }
57
58 #titles .title .word{
59     float: left;
60     text-align: center;
61     display: block;
62     /* border: 2px solid green; */
63     width: 250px;
64     margin: 0;
65     margin-left: 20px;
66     padding: 0;
67 }
68
69 #titles .title .buttons{
70     float: left;
71     width: 250px;
72     margin-left: 20px;
73     /* border: 2px solid orange; */
74 }
75
76 #titles .title .word p{
```

```

77  font-family:'fingerfunny';
78  font-size: 70pt;
79  text-align: center;
80  padding: 0;
81  margin: 0;
82 }
83
84 #startButton, #joinButton{
85  width: 100%;
86  height: 100px;
87  padding: 0;
88  border-radius: 30px;
89  border-width: 5pt;
90  border-color: #DB0A5B;
91  border-style:inset;
92  background-color: #F2784B;
93  font-family: "fingerfunny";
94  text-align: center;
95  font-size: 30pt;
96  color: white;
97 }
98
99 #startButton:hover , #joinButton:hover{
100  background-color: #F54707;
101 }
102
103 #titles .title object{
104  height:380px;
105  /* border: 2px solid purple; */
106 }
107
108 @media (max-width:1260px){
109  #left, #right, #startButton{
110   display: none;
111   float: none;
112 }
113
114  #joinButton{
115   visibility: visible;
116   width: 85%;
117 }
118 }
119
120 @media (max-width:800px){
121  #left, #right{
122   display: none;
123 }
124
125  #titles .title .word, #titles .title .buttons{
126   margin-left: -10px;
127 }
128
129  #titles .title .word p{
130   font-size: 55pt;
131 }
132 }
133
134 @media (min-width:1260px) {
135  #joinButton{
136   display: none;
137 }
138 }

```

## A.1.2 Kode Program Halaman Permintaan Bergabung

### A.1.2.1 Kode Program JavaScript Halaman Permintaan Bergabung pada PC

Listing A.5: syncScript.js

```

1 var socket = io();
2 var player = 0;
3 var players = [];
4 var text = 0;
5 var bg = $(".stage-area");
6
7 /**
8  * Socket.io connection related
9 */
10
11 socket.on('connect', function(){
12   getRandInt();
13 });
14
15 socket.on('requestAccepted', function(msg){
16   requestAccepted(msg);
17 });
18
19 socket.on('toCharPage', function(msg){
20   toCharDesk();
21 });
22
23 socket.on('selectingAcc', function(msg){
24   console.log(msg);
25 });
26
27 // Socket.io code related ends here

```

```

28
29 function toCharDesk(){
30   var charHtml = $("#charDesktop").html();
31   bg.html(charHtml);
32 }
33
34 function getRandInt(){
35   text = Math.floor(Math.random() * (999999 - 111111)) + 111111;
36   document.getElementById("roomId").innerHTML = text;
37
38   var roomString = text.toString();
39   console.log('this is a host room ${roomString}');
40
41   socket.emit('hostJoinRoom', {
42     id: socket.id,
43     room: roomString
44   });
45 }
46
47 function requestAccepted(msg){
48   var rom = text.toString();
49
50   //debug purpose
51   console.log('this is room: ${rom}');
52   console.log('sum of player: ${player}');
53
54   if (msg.room === rom) {
55     if (player == 0) {
56
57       //debug purpose
58       console.log('Player with id: ${msg.id} is joined');
59
60       var player1 = {
61         id: msg.id,
62         number: 1
63     }
64       players.push(player1);
65
66       var play1 = document.getElementById("player1");
67       play1.innerHTML = '<p>PLAYER 1 HAS JOIN THE GAME</p>';
68       player++;
69
70       //debug purpose
71       console.log('after player1 joining: ${player}');
72     }else {
73
74       //debug purpose
75       console.log('Player with id: ${msg.id} is joined');
76
77       var player2 = {
78         id: msg.id,
79         number: 2
80     }
81
82       players.push(player2);
83       console.log(players);
84
85       var play2 = document.getElementById("player2");
86       play2.innerHTML = '<p>PLAYER 2 HAS JOIN THE GAME</p>';
87       socket.emit('roomFull', {
88         // id: msg.id,
89         room: msg.room
90       });
91       player = 0;
92
93       console.log('after player2 joining: ${player}');
94     }
95   }
96 }

```

### A.1.2.2 Kode Program CSS Halaman Permintaan Bergabung pada *PC*

Listing A.6: syncStyle.css

```

1 /* body{
2   background-color: #F9690E;
3 } */
4
5 /* .bg{
6   width: 1210px;
7   height: 660px;
8   border-style: solid;
9   border-color: black;
10  border-width: thick;
11  margin-top: 20px;
12  margin-left: 20px;
13  background-color: #FDE3A7;
14 } */
15
16 .pageContainer{
17   width: 800px;
18   height: 400px;
19   margin: auto;
20   margin-top: 40px;
21 }
22
23 .step1{

```

```
24| width: 700px;
25| height: 180px;
26| border-style: solid;
27| border-color: black;
28| border-width: 3px;
29| border-radius: 15px;
30| margin: auto;
31| background-color: white;
32}
33
34.step1Title{
35| width: 300px;
36| height: 30px;
37| border-style: solid;
38| border-color: black;
39| border-width: 3px;
40| border-radius: 20px;
41| margin: auto;
42| margin-top: 5px;
43| background-color: black;
44}
45
46.step1Title h1{
47| font-size: 10pt;
48| text-align: center;
49| color: white;
50| font-family: sans-serif;
51}
52
53.step1 p{
54| margin: auto;
55| text-align: center;
56| margin-top: 10px;
57| font-family: sans-serif;
58}
59
60.link{
61| width: 350px;
62| height: 30px;
63| border-style: solid;
64| border-color: black;
65| border-width: 3px;
66| border-radius: 20px;
67| margin: auto;
68| margin-top: 20px;
69| background-color: #F5D76E;
70}
71
72.link p{
73| margin: auto;
74| text-align: center;
75| margin-top: 5px;
76| font-family: sans-serif;
77}
78
79.step2 {
80| width: 700px;
81| height: 180px;
82| border-style: solid;
83| border-color: black;
84| border-width: 3px;
85| border-radius: 15px;
86| margin: auto;
87| margin-top: 25px;
88| background-color: white;
89}
90
91.step2 p{
92| margin: auto;
93| text-align: center;
94| margin-top: 10px;
95| font-family: sans-serif;
96}
97
98.step2Title{
99| width: 300px;
100| height: 30px;
101| border-style: solid;
102| border-color: black;
103| border-width: 3px;
104| border-radius: 20px;
105| margin: auto;
106| margin-top: 5px;
107| background-color: black;
108}
109
110.step2Title h1{
111| font-size: 10pt;
112| text-align: center;
113| color: white;
114| font-family: sans-serif;
115}
116
117.wrapperCode{
118| width: 100px;
119| height: 35px;
120| border-style: solid;
121| border-color: black;
```

```

123 border-width: 3px;
124 border-radius: 20px;
125 margin: auto;
126 margin-top: 5px;
127 background-color: #F5D76E;
128 }
129
130 .wrapperCode p{
131   font-size: 12pt;
132   text-align: center;
133   color: black;
134   font-family: sans-serif;
135 }
136
137 #gameArea{
138   width: 500px;
139   height: 100px;
140   margin: auto;
141   margin-top: 10px;
142 }
143
144 .player{
145   width: 500px;
146   height: 100px;
147   margin: auto;
148   margin-top: 10px;
149
150   text-align: center;
151 }
152
153 .player1wrap{
154   width: 200px;
155   height: 70px;
156   margin: auto;
157   margin-top: 10px;
158   margin-left: 40px;
159   font-family: helvetica;
160   float: left
161 }
162
163 .player2wrap{
164   width: 200px;
165   height: 70px;
166   margin: auto;
167   margin-top: 10px;
168   margin-left: 15px;
169   font-family: helvetica;
170   float: left;
171 }
172
173 .player p{
174   font-family: 'fingerfunny';
175   font-size: 20pt;
176 }
```

### A.1.2.3 Kode Program JavaScript Halaman Permintaan Bergabung pada *Smartphone*

Listing A.7: mobileScript.js

```

1 //MAKE RESPONSIVE LAYOUT !!!
2
3 var socket = io();
4 // var startTime;
5 // Socket.io Code related
6
7 socket.on('connect', function(){
8   console.log('Client is connected !');
9   console.log('This is client id: ${socket.id}');
10 });
11
12 //When a client successfully join the room
13 //The message is shown here.
14 socket.on('joinSucceed', function(msg){
15   var messages = document.getElementById("joined");
16   messages.innerHTML = msg;
17   document.getElementById("requestButton").disabled = true;
18 });
19
20
21 //When the room is not exist, connection is rejected
22 socket.on('joinRejected', function(msg){
23   var messages = document.getElementById("joined");
24   messages.innerHTML = msg;
25 });
26
27 //When the room is full,
28 //Redirect to character page using toCharMobile() function
29 socket.on('toCharPage', function(msg){
30   toCharMobile();
31 });
32
33 // Socket.io Code Ends Here
34
35 //Client requesting to join the room
36 //After entering the code and click send
```

```

37| function requestToJoin(){
38|   var codeRoom = $('#code').val();
39|   if (codeRoom === "") {
40|     alert('Enter the code from the desktop');
41|   }else{
42|     socket.emit('requestToJoin', {
43|       id: socket.id,
44|       room: $('#code').val()
45|     });
46|   }
47|
48| // $('form').submit(function(e){
49| //   e.preventDefault();
50| //   socket.emit('requestToJoin', {
51| //     id: socket.id,
52| //     room: $('#code').val()
53| //   });
54| // });
55| }
56|
57//redirecting to a charMobile page.
58function toCharMobile(){
59  var charMobileHtml = $("#charMobile").html();
60  bg.html(charMobileHtml);
61}

```

#### A.1.2.4 Kode Program CSS Halaman Permintaan Bergabung pada *Smartphone*

Listing A.8: mobileStyle.css

```

1 /* body{
2   background-color: #F9690E;
3 } */
4
5 @font-face {
6   font-family: "fingerfunny";
7   src: url("Nickname_DEMO.otf");
8 }
9
10.formContainer {
11   text-align: center;
12   margin: 0 auto;
13 }
14
15.form{
16   text-align: center;
17   margin: 0 auto;
18 }
19
20#code{
21   margin-bottom: 25px;
22   width: 200px;
23   height: 30px;
24   font-family: helvetica;
25   border-style: solid;
26   border-radius: 10px;
27   font-size: 15pt;
28   background-color: #F5D76E;
29 }
30
31#requestButton{
32   font-family: fingerfunny;
33   font-size: 20pt;
34   width: 100px;
35   height: 50px;
36   text-align: center;
37   border-width: thick;
38   border-color: black;
39   background-color: #F5D76E;
40   border-radius: 7px;
41   text-align: center;
42 }
43
44#requestButton:hover, #requestButton:focus{
45   background-color: #DCB21D;
46 }
47
48#command, #joined{
49   text-align: center;
50   font-family: sans-serif;
51   font-size: 17pt;
52 }

```

#### A.1.3 Kode Program Halaman Memilih Karakter

##### A.1.3.1 Kode Program JavaScript Halaman Memilih Karakter pada *PC*

Listing A.9: charDesktopScript.js

```

1 //debug purpose begin
2 console.log('char desktop id is ${socket.id}');
3

```

```

4 socket.emit('charDesktop', 'Char Desktop is connecting to socket.io');
5
6 socket.on('charDesktopAcc', function(msg){
7   console.log(msg);
8 });
9 //debug purpose end
10
11 var marker = 0;
12 var playerData = [];
13 var timeoutID;
14
15 var imageArray = [];
16 imageArray.push('');
17 imageArray.push('');
18 imageArray.push('');
19 imageArray.push('');
20
21 //When a client is selecting a char on mobile page,
22 //The choosen character is shown.
23 socket.on('charSelecting', function(msg){
24   if (msg.id === players[0].id) {
25     var imagep1 = document.getElementById("image_1_char");
26     if (msg.val == 1) {
27       imagep1.innerHTML = imageArray[msg.val - 1];
28     }else if (msg.val == 2) {
29       imagep1.innerHTML = imageArray[msg.val - 1];
30     }else if (msg.val == 3) {
31       imagep1.innerHTML = imageArray[msg.val - 1];
32     }else{
33       imagep1.innerHTML = imageArray[msg.val - 1];
34     }
35   }else{
36     var imagep2 = document.getElementById("image_2_char");
37     if (msg.val == 1) {
38       imagep2.innerHTML = imageArray[msg.val - 1];
39     }else if (msg.val == 2) {
40       imagep2.innerHTML = imageArray[msg.val - 1];
41     }else if (msg.val == 3) {
42       imagep2.innerHTML = imageArray[msg.val - 1];
43     }else{
44       imagep2.innerHTML = imageArray[msg.val - 1];
45     }
46   }
47 });
48
49 socket.on('playerDisconnected', function(playerId){
50   if(!alert('Oh no, player has been disconnected! Find your partner to continue!')){
51     window.location.href = "/";
52   }
53 });
54
55 socket.on('charSent', function(msg){
56   marker = marker + msg.marker;
57   console.log('This is marker: ${marker}');
58
59   if (msg.id === players[0].id) {
60     var play1 = document.getElementById("char_1_name");
61     var playerData1 = {
62       playerId: msg.id,
63       charValue: msg.val
64     };
65     playerData[0] = playerData1;
66     console.log('this is value play 1 : ${msg.val}');
67     if (msg.val == 1) {
68       play1.innerHTML = 'Broco Dude';
69     }else if (msg.val == 2) {
70       play1.innerHTML = 'Dabu Donut';
71     }else if (msg.val == 3) {
72       play1.innerHTML = 'Grape Yoda';
73     }else{
74       play1.innerHTML = 'Summer Egg';
75     }
76   }else {
77     var play2 = document.getElementById("char_2_name");
78     var playerData2 = {
79       playerId: msg.id,
80       charValue: msg.val
81     };
82     playerData[1] = playerData2;
83     console.log('this is value play 2 : ${msg.val}');
84     if (msg.val == 1) {
85       play2.innerHTML = 'Broco Dude';
86     }else if (msg.val == 2) {
87       play2.innerHTML = 'Dabu Donut';
88     }else if (msg.val == 3) {
89       play2.innerHTML = 'Grape Yoda';
90     }else{
91       play2.innerHTML = 'Summer Egg';
92     }
93   }
94
95   if (marker == 2) {
96     marker = 0;
97     beginWaiting(1);
98
99     // toGamePlayDesktop();
100    // console.log('Player data: ${playerData[0].playerId}');
101    // socket.emit('charIsReady', playerData);
102  }

```

```

103 });
104
105 function beginWaiting(beginCounter){
106   var timer = setInterval(waitingToCall, 1000);
107
108   function waitingToCall(){
109     if (beginCounter >= 1) {
110       beginCounter -= 1;
111     }else{
112       clearInterval(timer);
113       toGamePlayDesktop();
114     }
115   }
116 }
117
118 function toGamePlayDesktop(){
119   var gamePlayDesktop = $("#gamePlayDesktop").html();
120   bg.html(gamePlayDesktop);
121   socket.emit('charIsReady', playerData);
122 }

```

### A.1.3.2 Kode Program CSS Halaman Memilih Karakter pada *PC*

Listing A.10: charDesktopStyle.css

```

1 .container_char_desktop{
2   width: 90%;
3   margin: auto;
4   overflow: hidden;
5   /* border: 2px solid red; */
6 }
7
8 .word_select_desktop{
9   text-align: center;
10  font-family: Arial;
11  margin: 10px;
12  /* border: 2px solid black; */
13 }
14
15 .player_char_container{
16   text-align: center;
17   width: 90%;
18   margin: auto;
19   padding: 10px;
20   margin-left: 7%;
21   /* border: 2px solid black; */
22 }
23
24 .player_char_container .player_1_char, .player_char_container .player_2_char{
25   /* border: 5px solid green; */
26   width: 40%;
27   float: left;
28   text-align: center;
29   margin: 10px;
30   padding: 20px;
31
32   border-radius: 30px;
33   border-width: 5pt;
34   border-color: #DB0A5B;
35   border-style: inset;
36 }
37
38 .player_1_char #image_1_char, .player_2_char #image_2_char{
39   text-align: center;
40   padding: 10px;
41   margin: 10px;
42   height: 300px;
43
44   /* border: 5px solid red; */
45 }
46
47 .player_1_char #image_1_char img, .player_2_char #image_2_char img{
48   height: 300px;
49 }
50
51 .player_1_char #char_1_name, .player_2_char #char_2_name{
52   text-align: center;
53   padding: 10px;
54   font-family: 'fingerfunny';
55   font-size: 40pt;
56
57   border-radius: 30px;
58   border-width: 5pt;
59   border-color: #DB0A5B;
60   background-color: #F2784B;
61   border-style: none;
62   /* border: 2px solid blue; */
63 }

```

### A.1.3.3 Kode Program JavaScript Halaman Memilih Karakter pada *Smartphone*

Listing A.11: charMobileScript.js

```
1 | var mark = 0;
```

```

2 // socket.io connection related starts here
3
4 socket.emit('charMobile', 'Char Mobile is connecting to Socket.io with id: ${socket.id}');
5
6 socket.on('CharMobileAccepted', function(msg){
7   console.log(msg);
8 });
9
10
11 socket.on('playerDisconnected', function(playerId){
12   if(!alert('Oh no, player has been disconnected! Find your partner to continue!')){
13     window.location.href = "/";
14   }
15 });
16
17 socket.on('charSent', function(msg){
18   mark += 1;
19   if (mark == 2) {
20     mark = 0;
21     waitForCall(1);
22     // toGamePlayMobile();
23   }
24 });
25
26 // socket.io connection related ends here
27
28 function waitForCall(beginCounter){
29   var timer = setInterval(countWaiting, 1000);
30
31   function countWaiting(){
32     if (beginCounter >= 1) {
33       beginCounter -= 1;
34     }else{
35       clearInterval(timer);
36       toGamePlayMobile();
37     }
38   }
39 }
40
41 function selectChar(){
42   var valButton = $('input[name="radioChar"]:checked').val();
43   socket.emit('selectingChar', {
44     val: valButton,
45     id: socket.id
46   });
47
48 //debug purpose
49 console.log('Player is choosing char number ${valButton}');
50 }
51
52 function toGamePlayMobile(){
53   var gpMobile = $('#gamePlayMobile').html();
54   bg.html(gpMobile);
55 }
56
57 function sendChar(){
58   var valButton = $('input[name="radioChar"]:checked').val();
59   console.log(valButton);
60   if (valButton === undefined) {
61     alert("You have not choose a character.");
62   }
63   else{
64     socket.emit('sendingChar', {
65       val: valButton,
66       id: socket.id,
67       marker: 1
68     });
69     document.getElementById("nextButton").disabled = true;
70   }
71 }

```

#### A.1.3.4 Kode Program CSS Halaman Memilih Karakter pada *Smartphone*

Listing A.12: charMobileStyle.css

```

1 .container_char{
2   width: 90%;
3   margin: auto;
4   overflow: hidden;
5   /* border: 2px solid red; */
6 }
7
8 .word_select{
9   font-family: Arial;
10  text-align: center;
11  /* border: 2px solid blue; */
12 }
13
14 .charList{
15   text-align: center;
16   padding: 20px;
17   /* border: 2px solid black; */
18 }
19
20 .charList input[type=radio]{
21   display: none;
22 }

```

```

23
24 input[type=radio]:checked + label img{
25   border-radius: 30px;
26   border-width: 5pt;
27   border-color: #DB0A5B;
28   border-style: inset;
29   background-color: #F2784B;
30   padding: 5px;
31 }
32
33 .charList label{
34   padding: 20px;
35 }
36
37 .charList label img{
38   height: 200px;
39 }
40
41 .send_button{
42   padding: 20px;
43   text-align: center;
44   /* border: 2px solid green; */
45 }
46
47 .send_button button{
48   width: 100%;
49   height: 70px;
50   padding: 0;
51   border-radius: 30px;
52   border-width: 5pt;
53   border-color: #DB0A5B;
54   border-style: inset;
55   background-color: #F2784B;
56   font-family: "fingerfunny";
57   text-align: center;
58   font-size: 30pt;
59   color: white;
60 }
61
62 .send_button button:hover, .send_button button:focus{
63   background-color: #F54707;
64 }

```

#### A.1.4 Kode Program Halaman Memulai Permainan

##### A.1.4.1 Kode Program JavaScript Halaman Memulai Permainan pada PC

Listing A.13: gamePlayDesktopScript.js

```

1 var canvas = document.getElementById('canvasGpDesktop');
2 var ctx = canvas.getContext('2d');
3 var ctx2 = canvas.getContext('2d');
4
5 var track = new Image();
6 var timer1 = new Image();
7 var timer2 = new Image();
8 var timer3 = new Image();
9
10 var player1Char = new Image();
11 var player2Char = new Image();
12
13 var data0fPlayer;
14
15 var player1Val = 0;
16 var player2Val = 0;
17
18 var winner = 0;
19
20 var timerArray = [];
21 var charArray = [];
22
23 var arr0fPlayerChar = [2];
24
25 var aniFrame;
26 var aniFrame2;
27 var progressPlayer1 = 0;
28 var progressPlayer2 = 0;
29 var xPosition1 = 100;
30 var yPosition1 = 340;
31 var xPosition2 = 90;
32 var yPosition2 = 400;
33
34 var counterLatencyP1 = 0;
35 var counterLatencyP2 = 0;
36 var totalLatencyP1 = 0;
37 var totalLatencyP2 = 0;
38
39 timer1.src = 'images/timer1.png';
40 timer2.src = 'images/timer2.png';
41 timer3.src = 'images/timer3.png';
42
43 charArray.push('images/brocoDudeTiny.png');
44 charArray.push('images/dabuDonutTiny.png');
45 charArray.push('images/grapeYodaTiny.png');
46 charArray.push('images/summerEggTiny.png');
47

```

```
48 timerArray.push(timer1);
49 timerArray.push(timer2);
50 timerArray.push(timer3);
51
52 socket.on('moveThePlayer', function(msg){
53   if (msg === dataOfPlayer[0].playerId) {
54     socket.on('pong', function(startTime){
55       var times = new Date();
56       var currentTime = times.getMilliseconds();
57       var latency = currentTime - startTime;
58       totalLatencyP1 += latency;
59       counterLatencyP1 += 1;
60     });
61     readyPlayerOne();
62   }else {
63     socket.on('pong', function(startTime){
64       var times = new Date();
65       var currentTime = times.getMilliseconds();
66       var latency = currentTime - startTime;
67       totalLatencyP2 += latency;
68       counterLatencyP2 += 1;
69     });
70     readyPlayerTwo();
71   }
72 });
73 });
74
75 socket.on('playerDisconnected', function(playerId){
76   if(!alert('Oh no, player has been disconnected! Find your partner to continue!')){
77     window.location.href = "/";
78   }
79 });
80
81 socket.on('startTheGame', function(msg){
82   init();
83
84   //debug purpose
85   dataOfPlayer = msg;
86   console.log(dataOfPlayer);
87   console.log('data of player 1: ${dataOfPlayer[0].playerId}');
88   console.log('data of player 2: ${dataOfPlayer[1].playerId}');
89   // ends here
90
91   beginCountDown(3, msg);
92 });
93
94 socket.on('toWinnerPage', function(msg){
95   toWinningPage();
96 });
97
98 function init(){
99   track.src = 'images/track.png';
100  track.onload = function(){
101    ctx.drawImage(track, 0, 0);
102  }
103 }
104
105 function beginCountDown(beginCounter, msg){
106   var counter = beginCounter - 1;
107   console.log('before showCountDown: ${counter}');
108   var timer = setInterval(showCountDown, 1000);
109
110   function showCountDown(){
111     if (beginCounter >= 1) {
112       ctx.drawImage(timerArray[counter], 330, 150);
113       beginCounter -= 1;
114       counter -= 1;
115       console.log('after showCountDown: ${counter}');
116     }else{
117       clearInterval(timer);
118       drawChar(msg);
119       beginGamePlay();
120     }
121   }
122 }
123
124 function drawChar(data){
125   player1Val = data[0].charValue;
126   player2Val = data[1].charValue;
127
128   console.log('this is player1Val : ${player1Val}');
129   console.log('this is player2Val : ${player2Val}');
130
131   player1Char.src = charArray[player1Val - 1];
132   player2Char.src = charArray[player2Val - 1];
133
134   arrOfPlayerChar[0] = player1Char;
135   arrOfPlayerChar[1] = player2Char;
136 }
137
138 function beginGamePlay(){
139   readyPlayerOne();
140   readyPlayerTwo();
141 }
142
143 function reachFinishLine(player){
144   if (winner == 0) {
145     winner = player;
146   }
```

```
147     socket.emit('sendingTheWinner',{
148         playerWin: winner,
149         playerCharArr: arrOfPlayerChar
150     });
151     // toWinningPage();
152 }
153
154 function toWinningPage(){
155     var winDesktop = $('#winningDesktop').html();
156     bg.html(winDesktop);
157 }
158
159 function readyPlayerOne(){
160     ctx.globalCompositeOperation = 'destination-over';
161     ctx.clearRect(0, 0, 900, 600);
162
163     ctx.save();
164     console.log('lets draw player 1');
165
166     progressPlayer1 += 1;
167     if ( xPosition1 < 680 && yPosition1 == 340 ) {
168         //BOTTOM ROW
169         console.log('bottom Row');
170         ctx.save();
171         xPosition1 += 0.5;
172         ctx.drawImage(player2Char, xPosition2, yPosition2);
173         ctx.drawImage(player1Char, xPosition1, yPosition1);
174         ctx.restore();
175     }else if(yPosition1 > 250){
176         console.log('right curve');
177         // RIGHT CURVE
178
179         ctx.save();
180
181         console.log('xPosition1: ${xPosition1}');
182         console.log('yPosition1: ${yPosition1}');
183
184         if(yPosition1 > 300) {
185             xPosition1 += 0.2;
186             yPosition1 -= 0.5;
187         }else{
188             xPosition1 -= 0.3;
189             yPosition1 -= 0.5;
190         }
191
192         ctx.drawImage(player1Char, xPosition1, yPosition1);
193         ctx.drawImage(player2Char, xPosition2, yPosition2);
194         ctx.restore();
195
196     }else if(xPosition1 > 150 && yPosition1 >= 250){
197         console.log('middle row');
198         // MIDDLE ROW
199
200         xPosition1 -= 0.5;
201         ctx.save();
202
203         ctx.drawImage(player1Char, xPosition1, yPosition1);
204         ctx.drawImage(player2Char, xPosition2, yPosition2);
205         ctx.restore();
206     }else if(xPosition1 < 680 && yPosition1 >= 10){
207         // LEFT CURVE
208         console.log('left curve');
209         ctx.save();
210
211         if(yPosition1 > 120) {
212             if (yPosition1 > 160) {
213                 xPosition1 -= 0.7;
214                 yPosition1 -= 0.6;
215             }else {
216                 yPosition1 -= 0.5;
217             }
218         }else{
219             xPosition1 += 0.5;
220             yPosition1 -= 0.7;
221         }
222
223         ctx.drawImage(player2Char, xPosition2, yPosition2);
224         ctx.drawImage(player1Char, xPosition1, yPosition1);
225         ctx.restore();
226
227     }else if(xPosition1 < 700 && yPosition1 >= 8){
228         //TOP ROW
229         console.log('top row');
230         ctx.save();
231         xPosition1 += 0.5;
232         ctx.drawImage(player2Char, xPosition2, yPosition2);
233         ctx.drawImage(player1Char, xPosition1, yPosition1);
234         ctx.restore();
235     }else {
236         reachFinishline(1);
237         ctx.drawImage(player2Char, xPosition2, yPosition2);
238         ctx.drawImage(player1Char, xPosition1, yPosition1);
239     }
240
241     ctx.restore();
242
243     ctx.drawImage(track, 0, 0);
244
245     if (progressPlayer1 < 40) {
        console.log('progressPlayer1: ${progressPlayer1}');
```

```
246     aniFrame = requestAnimationFrame(readyPlayerOne);
247 }
248 else {
249     progressPlayer1 = 0;
250 }
251 }
252 // USE ARRAY FOR PLAYER AND CHARACTER
253
254 function readyPlayerTwo(){
255     ctx.globalCompositeOperation = 'destination-over';
256     ctx.clearRect(0, 0, 900, 600);
257
258     ctx.save();
259     progressPlayer2 += 1;
260
261     if (xPosition2 < 680 && yPosition2 == 400 ) {
262         console.log('bottom Row 2');
263         xPosition2 += 0.5;
264         ctx.drawImage(player2Char, xPosition2, yPosition2);
265         ctx.drawImage(player1Char, xPosition1, yPosition1);
266     }else if(yPosition2 > 190){
267         console.log('right curve 2');
268         // RIGHT CURVE
269
270         ctx.save();
271
272         console.log('xPosition2: ${xPosition2}');
273         console.log('yPosition2: ${yPosition2}');
274
275         if(yPosition2 > 250) {
276             if (yPosition2 > 340) {
277                 xPosition2 += 0.7;
278                 yPosition2 -= 0.6;
279             }else {
280                 yPosition2 -= 0.5;
281             }
282         }else{
283             xPosition2 -= 0.5;
284             yPosition2 -= 0.7;
285         }
286
287         ctx.drawImage(player2Char, xPosition2, yPosition2);
288         ctx.drawImage(player1Char, xPosition1, yPosition1);
289
290         ctx.restore();
291
292     }else if(xPosition2 > 150 && yPosition2 >= 180){
293         console.log('middle row 2');
294         // MIDDLE ROW
295
296         xPosition2 -= 0.5;
297         ctx.save();
298
299         ctx.drawImage(player1Char, xPosition1, yPosition1);
300         ctx.drawImage(player2Char, xPosition2, yPosition2);
301         ctx.restore();
302     }else if(xPosition2 < 680 && yPosition2 >= 70){
303         // LEFT CURVE
304         console.log('left curve 2');
305
306         ctx.save();
307
308         if(yPosition2 > 100) {
309             xPosition2 -= 0.2;
310             yPosition2 -= 0.5;
311         }else{
312             xPosition2 += 0.3;
313             yPosition2 -= 0.5;
314         }
315
316         ctx.drawImage(player2Char, xPosition2, yPosition2);
317         ctx.drawImage(player1Char, xPosition1, yPosition1);
318         ctx.restore();
319     }else if(xPosition2 < 700 && yPosition2 >= 65){
320         console.log('top row 2');
321         ctx.save();
322         xPosition2 += 0.5;
323         ctx.drawImage(player2Char, xPosition2, yPosition2);
324         ctx.drawImage(player1Char, xPosition1, yPosition1);
325         ctx.restore();
326     }else {
327         reachFinishLine(2);
328         ctx.drawImage(player2Char, xPosition2, yPosition2);
329         ctx.drawImage(player1Char, xPosition1, yPosition1);
330     }
331
332     ctx.restore();
333
334     ctx.drawImage(track, 0, 0);
335
336     if (progressPlayer2 < 40) {
337         console.log('progressPlayer2: ${progressPlayer2}');
338         aniFrame2 = requestAnimationFrame(readyPlayerTwo);
339     }
340     else {
341         progressPlayer2 = 0;
342     }
343 }
```

#### A.1.4.2 Kode Program CSS Halaman Memilih Karakter pada *PC*

Listing A.14: gamePlayDesktopStyle.css

```

1 .canvasContainer{
2     /* padding: 10px;
3     padding-left: 140px; */
4     margin: auto;
5     text-align: center;
6 }
7
8 .stage-area{
9     background-color: #26C281;
10 }
```

#### A.1.4.3 Kode Program JavaScript Halaman Memulai Permainan pada *Smartphone*

Listing A.15: gamePlayMobileScript.js

```

1 var stepLeftHtml = document.getElementById('stepLeft');
2 var instructionEl = document.getElementById('instruction');
3
4 socket.on('startTheGame', function(msg){
5     showInstruction(3);
6 });
7
8 socket.on('toWinnerPage', function(msg){
9     moveToWinPage();
10 });
11
12 socket.on('playerDisconnected', function(playerId){
13     if(!alert('Oh no, player has been disconnected! Find your partner to continue!')){
14         window.location.href = "/";
15     }
16 });
17
18
19 function stepClicked(){
20     socket.emit('stepClicked', {
21         playerID: socket.id
22     });
23
24     var time = new Date();
25     startTime = time.getMilliseconds();
26     socket.emit('ping', startTime);
27 }
28
29 function moveToWinPage(){
30     var winMobile= $('#winningMobile').html();
31     bg.html(winMobile);
32 }
33
34
35 function showInstruction(beginCounter){
36     instructionEl.innerHTML = 'Press left and right to move your character';
37
38     var timer = setInterval(showText, 1000);
39
40     function showText(){
41         beginCounter -= 1;
42         if (beginCounter < 1) {
43             instructionEl.innerHTML = '';
44             clearInterval(timer);
45         }
46     }
47 }
```

#### A.1.4.4 Kode Program CSS Halaman Memulai Permainan pada *Smartphone*

Listing A.16: gamePlayMobileStyle.css

```

1 /* #stepButton{
2     background: url('images/stepUp.png') 100px 100px no-repeat;
3 } */
4 #instruction{
5     font-size: 20pt;
6     font-family: Arial;
7     text-align: center;
8 }
9
10 .step_container{
11     text-align: center;
12     margin-top: 150px;
13     margin-bottom: 150px;
14 }
15
16 #step_img{
17     height: 200px;
18 }
19
20 #step{
21     padding: 20px;
22 }
```

### A.1.5 Kode Program Halaman Mengakhiri Permainan

#### A.1.5.1 Kode Program JavaScript Halaman Mengakhiri Permainan pada *PC*

Listing A.17: winningDesktopScript.js

```

1 var winCanvas = document.getElementById('canvasWinStage');
2 var winCtx = winCanvas.getContext('2d');
3
4 var podium = new Image();
5
6 var winnerCharArr = [];
7
8 winnerCharArr.push('images/brocoDudeTiny.png');
9 winnerCharArr.push('images/dabuDonutTiny.png');
10 winnerCharArr.push('images/grapeYodaTiny.png');
11 winnerCharArr.push('images/summerEggTiny.png');
12
13 socket.on('getTheWinner', function(msg){
14     var averageLatencyP1 = totalLatencyP1 / counterLatencyP1;
15     var averageLatencyP2 = totalLatencyP2 / counterLatencyP2;
16
17     console.log('This is counter latency p1: ${counterLatencyP1}');
18     console.log('This is counter latency p2: ${counterLatencyP2}');
19
20     console.log('This is average latency p1: ${averageLatencyP1}');
21     console.log('This is average latency p2: ${averageLatencyP2}');
22
23     console.log('The Winner is : ${msg.playerWin}');
24     drawWinner(msg.playerWin);
25 });
26
27
28 socket.on('backHome', function(msg){
29     var homePage = $("#homePage").html();
30     bg.html(homePage);
31 });
32
33 function drawStage(){
34     podium.src = 'images/winning.png';
35     podium.onload = function(){
36         winCtx.drawImage(podium, 0, 0);
37     }
38 }
39
40 function drawWinner(winner){
41     if (winner == 1) {
42         drawFirstWinner(player1Val);
43         drawSecondWinner(player2Val);
44     }else{
45         drawFirstWinner(player2Val);
46         drawSecondWinner(player1Val);
47     }
48 }
49
50 function drawFirstWinner(firstWinner){
51     var first = new Image();
52     first.src = winnerCharArr[firstWinner - 1];
53     winCtx.drawImage(first, 40, 210);
54 }
55
56 function drawSecondWinner(secondWinner){
57     var second = new Image();
58     second.src = winnerCharArr[secondWinner - 1];
59     winCtx.drawImage(second, 180, 330);
60 }
61
62 function toHomePage(){
63     socket.emit('goBackHome', 'back home');
64 }
65 drawStage();

```

#### A.1.5.2 Kode Program CSS Halaman Mengakhiri Permainan pada *PC*

Listing A.18: winningDesktopStyle.css

```

1 .canvasWinStageContainer{
2     margin: auto;
3 }
4
5 #exitButton{
6     width: 250px;
7     height: 80px;
8
9     border-radius: 30px;
10    border-width: 5pt;
11    border-color: #DB0A5B;
12    border-style:inset;
13    background-color: #F2784B;
14
15    font-family: "fingerfunny";
16    text-align: center;
17
18    font-size: 30pt;

```

```

19|   color: white;
20|
21|
22#exitButton:hover{
23  background-color: #F54707;
24}
25
26.stage-area{
27  background-color: #26C281;
28  padding-left: 400px;
29}

```

### A.1.5.3 Kode Program JavaScript Halaman Mengakhiri Permainan pada *Smartphone*

Listing A.19: winningMobileScript.js

```

1 var game_over = document.getElementById('gameOver');
2
3 socket.on('getTheWinner', function(msg){
4   showGameOver(msg.playerWin);
5 });
6
7 socket.on('backHome', function(msg){
8   backToHome();
9 });
10
11 function showGameOver(winner){
12   gameOver.innerHTML = '<p>GAME OVER</p>';
13 }
14
15 function backToHome(){
16   var HomePage = $('#HomePage').html();
17   bg.html(HomePage);
18 }

```

### A.1.5.4 Kode Program CSS Halaman Mengakhiri Permainan pada *Smartphone*

Listing A.20: charMobileStyle.css

```

1 .gameOverContainer{
2   margin: auto;
3   padding: 10px;
4 }
5
6 .gameOverContainer #gameOver{
7   font-family: 'fingerfunny';
8   font-size: 80pt;
9   text-align: center;
10 }

```

## A.2 Kode Program Direktori *routes*

### A.2.1 Kode Program *homeRoutes.js*

Listing A.21: homeRoutes.js

```

1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('home', { title: 'Home Pages' });
7 });
8
9 module.exports = router;

```

## A.3 Kode Program Direktori *views*

### A.3.1 Kode Program Halaman *Error*

Listing A.22: error.ejs

```

1 <h1><%= message %></h1>
2 <h2><%= error.status %></h2>
3 <pre><%= error.stack %></pre>

```

### A.3.2 Kode Program Seluruh Halaman pada Finger For Life

Listing A.23: home.ejs

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8" >
5          <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
6          user-scalable=0, shrink-to-fit=no">
7
8          <title>Finger For Life</title>
9          <link rel="stylesheet" href="stylesheets/homeStyle.css">
10         </head>
11         <body>
12             <div class="stage-area">
13
14             </div>
15
16             <template id="homePage">
17                 <section id="titles">
18                     <div class="container">
19                         <div class="title_container">
20                             <div class="title" id="left">
21                                 <object type="image/svg+xml" data="images/finga.svg"></object>
22                             </div>
23
24                             <div class="title">
25                                 <div class="word">
26                                     <p>Finger <br> For <br> Life</p>
27                                 </div>
28                             <div class="buttons">
29                                 <button id="startButton" onclick="startClicked()">START</button>
30                                 <button id="joinButton" onclick="joinClicked()">JOIN</button>
31                             </div>
32                         </div>
33
34                         <div class="title" id="right">
35                             <object type="image/svg+xml" data="images/finga.svg"></object>
36                         </div>
37                     </div>
38                 </section>
39             </template>
40
41             <!-- Mobile Page -->
42             <template id="mobilePage">
43                 <link rel="stylesheet" href="stylesheets/mobileStyle.css">
44                 <div class="formContainer">
45                     <p id="command">Enter the code from the desktop below:</p>
46                     <!-- <form> -->
47                     <input id="code" type="number" autocomplete="off" autofocus> <br>
48                     <button id="requestButton" onclick="requestToJoin()">SEND</button>
49                     <!-- </form> -->
50                 </div>
51
52                 <p id="joined"></p>
53
54             <!-- Character Page Mobile -->
55             <template id="charMobile">
56                 <link rel="stylesheet" href="stylesheets/charMobileStyle.css">
57                 <section id="char_section">
58                     <div class="container_char">
59                         <div class="word_select">
60                             <h1>Please select a character and press "choose" to play the game.</h1>
61                         </div>
62
63                         <div class="charList">
64
65                             <input type="radio" name="radioChar" id="char1" value="1" onclick="selectChar()">
66                             <label for="char1"></label>
67
68                             <input type="radio" name="radioChar" id="char2" value="2" onclick="selectChar()">
69                             <label for="char2"></label>
70
71                             <input type="radio" name="radioChar" id="char3" value="3" onclick="selectChar()">
72                             <label for="char3"></label>
73
74                             <input type="radio" name="radioChar" id="char4" value="4" onclick="selectChar()">
75                             <label for="char4"></label>
76
77                         </div>
78
79                         <div class="send_button">
80                             <button onclick="sendChar()" id="nextButton">CHOOSE</button>
81                         </div>
82                     </div>
83                 </section>
84                 <script src="/javascripts/charMobileScript.js"></script>
85
86             <!-- GamePlay Mobile Page -->
87             <template id="gamePlayMobile">
88                 <link rel="stylesheet" href="stylesheets/gamePlayMobileStyle.css">
89                 <div class="gameplay_container">
90                     <div id="instruction">
91
92                     </div>
93
94                     <div class="step_container">
95                         <span id="step" onclick="stepClicked()">
96                             
97                         </span>

```

```
98
99      <span id="step" onclick="stepClicked()">
100        
101      </span>
102    </div>
103
104  </div>
105
106  <template id="winningMobile">
107    <link rel="stylesheet" href="stylesheets/winningMobileStyle.css">
108    <div class="gameOverContainer">
109      <div id="gameOver">
110
111        </div>
112      </div>
113
114      <script src="/javascripts/winningMobileScript.js"></script>
115    </template>
116
117    <script src="/javascripts/gamePlayMobileScript.js"></script>
118  </template>
119
120 </template>
121
122 <script src="/javascripts/mobileScript.js"></script>
123 </template>
124 <!-- Mobile Page ends here -->
125
126 <!-- Sync Page -->
127 <template id="syncPage">
128   <link rel="stylesheet" href="stylesheets/syncStyle.css">
129   <div class="pageContainer">
130     <div class="step1">
131       <div class="step1Title">
132         <h1>SYNC STEP 1</h1>
133       </div>
134       <p>Enter the link below in your smartphone or tablet browser.<br>
135         click join to play the game.</p>
136       <div class="link">
137         <p>fingerforlife.herokuapp.com</p>
138       </div>
139     </div>
140
141     <div class="step2">
142       <div class="step2Title">
143         <h1>SYNC STEP 2</h1>
144       </div>
145       <p>Enter the code below<br> on the mobile site.</p>
146       <div class="wrapperCode">
147         <p id="roomid"></p>
148       </div>
149     </div>
150   </div>
151
152   <div class="player">
153     <div class="player1wrap">
154       <div id="player1"></div>
155     </div>
156     <div class="player2wrap">
157       <div id="player2"></div>
158     </div>
159
160   </div>
161
162 <!-- Choose a Char Page Desktop -->
163 <template id="charDesktop">
164   <link rel="stylesheet" href="stylesheets/charDesktopStyle.css">
165   <section id="char_desktop_section">
166     <div class="container_char_desktop">
167       <div class="word_select_desktop">
168         <h1>Choose a character on your mobile site.</h1>
169       </div>
170
171       <div class="player_char_container">
172         <div class="player_1_char">
173           <div id="image_1_char">
174
175             </div>
176
177           <div id="char_1_name">
178
179             </div>
180           </div>
181
182         <div class="player_2_char">
183           <div id="image_2_char">
184
185             </div>
186
187           <div id="char_2_name">
188
189             </div>
190           </div>
191         </div>
192       </div>
193     </section>
194
195   <!-- Game Play Page on Desktop -->
196   <template id="gamePlayDesktop">
```

```

197     <link rel="stylesheet" href="stylesheets/gamePlayDesktopStyle.css">
198
199     <div class="canvasContainer">
200         <canvas id="canvasGpDesktop" width="900" height="600"></canvas>
201     </div>
202
203     <script src="/javascripts/gamePlayDesktopScript.js"></script>
204
205     <!-- Winning Page on Desktop -->
206     <template id="winningDesktop">
207         <link rel="stylesheet" href="stylesheets/winningDesktopStyle.css">
208
209
210         <span class="canvasWinStageContainer">
211             <canvas id="canvasWinStage" width="400" height="600"></canvas>
212         </span>
213
214         <span class="navigatorContainer">
215             <button id="exitButton" onclick="toHomePage()">Exit</button>
216         </span>
217
218
219         <script src="/javascripts/winningDesktopScript.js"></script>
220     </template>
221
222     </template>
223
224     <script src="/javascripts/charDesktopScript.js"></script>
225 </template>
226
227     <script src="/javascripts/syncScript.js"></script>
228 </template>
229
230     <script src="/socket.io/socket.io.js"></script>
231     <script src="javascripts/libs/jquery-3.3.1.min.js"></script>
232     <script src="javascripts/homeScript.js"></script>
233
234 </body>
235
236 </html>

```

## A.4 Kode Program *app.js*

Listing A.24: *app.js*

```

1 var createError = require('http-errors');
2 var express = require('express');
3 var path = require('path');
4 var cookieParser = require('cookie-parser');
5 var logger = require('morgan');
6
7 var homeRouter = require('../routes/homeRoutes');
8
9 var app = express();
10
11 // view engine setup
12 app.set('views', path.join(__dirname, 'views'));
13 app.set('view engine', 'ejs');
14
15 app.use(logger('dev'));
16 //app.use(express.json());
17 //app.use(express.urlencoded({ extended: false }));
18 app.use(cookieParser());
19 app.use(express.static(path.join(__dirname, 'public')));
20
21 app.use('/', homeRouter);
22
23 // catch 404 and forward to error handler
24 // app.use(function(req, res, next) {
25 //   next(createError(404));
26 // });
27
28 // error handler
29 // app.use(function(err, req, res, next) {
30 //   // set locals, only providing error in development
31 //   res.locals.message = err.message;
32 //   res.locals.error = req.app.get('env') === 'development' ? err : {};
33 //
34 //   // render the error page
35 //   res.status(err.status || 500);
36 //   res.render('error');
37 // });
38
39 module.exports = app;

```



## LAMPIRAN B

### KODE PROGRAM *SERVER*

#### B.1 Kode Program Kelas *users.js*

Listing B.1: *users.js*

```
1| class Users{
2|   constructor(){
3|     this.userArray = [];
4|   }
5|
6|   addUser(id, room){
7|     var newUser = {id, room};
8|     this.userArray.push(newUser);
9|     console.log(this.userArray);
10|    return newUser;
11|  }
12|
13|  isRoomExist(room){
14|    var result = false;
15|    console.log('The room to be Checked: ${room}');
16|    var tempUser = this.userArray.filter((user) => user.room === room);
17|    console.log(tempUser);
18|    if (tempUser.length == 0) {
19|      result = false;
20|      console.log('result false: ${result}');
21|    }else{
22|      result = true;
23|      console.log('result true: ${result}');
24|    }
25|
26|    return result;
27|  }
28|
29|  getUser(id){
30|    return this.userArray.filter((user) => user.id === id)[0];
31|  }
32|
33|  removeUser(id){
34|    var tempUser = this.userArray.filter((user) => user.id === id);
35|
36|    if (tempUser) {
37|      this.userArray = this.userArray.filter((user) => user.id !== id);
38|    }
39|
40|    return tempUser;
41|  }
42|
43|  removeRoom(room){
44|    var tempRoom = this.userArray.filter((user) => user.room === room);
45|
46|    if (tempRoom) {
47|      this.userArray = this.userArray.filter((user) => user.room !== room);
48|    }
49|
50|    return tempRoom;
51|  }
52|
53|  getUserList(room){
54|    var userList = this.userArray.filter((user) => user.room === room);
55|    var idList = userList.map((user) => user.id);
56|
57|    return idList;
58|  }
59| }
60|
61| module.exports = {Users};
```

#### B.2 Kode Program *www*

```
1|#!/usr/bin/env node
2|
3|/**
```

```

4  * Module dependencies.
5  */
6
7 var app = require('../app');
8 var http = require('http');
9 var socketIO = require('socket.io');
10 var {Users} = require('./utils/users');
11
12 /**
13  * Get port from environment and store in Express.
14 */
15 var port = normalizePort(process.env.PORT || '3000');
16 app.set('port', port);
17
18 /**
19  * Create HTTP server.
20 */
21 var server = http.createServer(app);
22
23 /**
24  * Create Socket.io Server
25 */
26 var io = socketIO(server);
27
28 /**
29  * Create Users Object
30 */
31 var users = new Users();
32
33 /**
34  * Listen to socket.io connection
35 */
36 io.on('connection', function(socket){
37
38   socket.on('hostJoinRoom', (msg) => {
39     socket.join(msg.room);
40     users.removeUser(msg.id);
41     users.addUser(msg.id, msg.room);
42   }),
43
44   socket.on('sendingTheWinner', (winner) => {
45     var user = users.getUser(socket.id);
46     io.to(user.room).emit('getTheWinner', winner);
47
48     io.to(user.room).emit('toWinnerPage', winner);
49   }),
50
51   socket.on('goBackHome', (msg) => {
52     var user = users.getUser(socket.id);
53     users.removeRoom(user.room);
54     io.to(user.room).emit('backHome', msg);
55   }),
56
57   socket.on('ping', (message) => {
58     socket.emit('pong', message);
59   }),
60
61 //When a room is already full
62 //redirecting to the character page
63 socket.on('roomFull', (msg) => {
64   io.to(msg.room).emit('toCharPage');
65 }), 
66
67 //When a user is selecting a character
68 //And the character which is clicked is shown.
69 socket.on('selectingChar', (msg) => {
70   var user = users.getUser(socket.id);
71   io.to(user.room).emit('charSelecting', msg);
72 }), 
73
74 socket.on('sendingChar', (msg) => {
75   var user = users.getUser(socket.id);
76   io.to(user.room).emit('charSent', msg);
77 }), 
78
79 socket.on('charIsReady', (msg) => {
80   var user = users.getUser(socket.id);
81   io.to(user.room).emit('startTheGame', msg);
82 }), 
83
84 socket.on('stepClicked', (message) => {
85   var user = users.getUser(socket.id);
86   io.to(user.room).emit('moveThePlayer', socket.id);
87 }), 
88
89 //debug purpose
90 socket.on('charMobile', (msg) => {
91   socket.emit('CharMobileAccepted', msg);
92 }), 
93
94 // When a user requesting to join the game
95 socket.on('requestToJoin', function(msg){
96   var check = users.isRoomExist(msg.room);
97   var roomList = users.getUserList(msg.room);
98
99   if (check === true) {
100     if (roomList.length === 3) {
101       socket.emit('joinRejected', 'The room is already full');
102     }
103   }
104 })

```

```
103    }
104    else{
105        socket.join(msg.room);
106        users.removeUser(msg.id);
107        users.addUser(msg.id, msg.room);
108
109        io.to(user.room).emit('requestAccepted', msg);
110        socket.emit('joinSucceed', 'Welcome to the game !');
111    }
112
113    }else{
114        socket.emit('joinRejected', 'The room is not exist');
115    }
116 });
117 });
118
119 /**
120 * Listen on provided port, on all network interfaces.
121 */
122
123 server.listen(port, (req, res) => {
124     console.log("Listening to " + port);
125 });
126
127
128 /**
129 * Normalize a port into a number, string, or false.
130 */
131
132 function normalizePort(val) {
133     var port = parseInt(val, 10);
134
135     if (isNaN(port)) {
136         // named pipe
137         return val;
138     }
139
140     if (port >= 0) {
141         // port number
142         return port;
143     }
144
145     return false;
146 }
```