TUGAS AKHIR - KI141502

RANCANG BANGUN APLIKASI AKTIVITAS LIVE CODING PADA PEMROGRAMAN WEB SECARA *PEER-TO-PEER* PADA PERAMBAN BERBASIS WEBRTC

MUHAMAD ROHMAN
NRP 5111100043

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D

Dosen Pembimbing II
Ir. Muchammad Husni, M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

UNDERGRADUATE THESIS - KI141502

DESIGN AND IMPLEMENTATION OF LIVE CODING ACTIVITY APPLICATION IN WEB PROGRAMMING USING PEER-TO-PEER IN BROWSER BASED ON WEBRTC

MUHAMAD ROHMAN
NRP 5111100043

Supervisor I
Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D

Supervisor II
Ir. Muchammad Husni, M.Kom.

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI AKTIVITAS LIVE CODING PADA PEMROGRAMAN WEB SECARA *PEER-TO-PEER* PADA PERAMBAN BERBASIS WEBRTC

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Muhamad Rohman
NRP: 5111100043

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.Eng. .....
NIP: 197708242006041001 (Pembimbing 1)

Ir. Muchammad Husni, M.Kom. .....
NIP: 196002211984031001 (Pembimbing 2)

SURABAYA
Juni 2016

RANCANG BANGUN APLIKASI AKTIVITAS LIVE CODING PADA PEMROGRAMAN WEB SECARA *PEER-TO-PEER* PADA PERAMBAN BERBASIS WEBRTC

Nama	:	**MUHAMAD ROHMAN**
NRP	:	**5111100043**
Jurusan	:	**Teknik Informatika FTIf**
Pembimbing I	:	**Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D**
Pembimbing II	:	**Ir. Muchammad Husni, M.Kom.**

Abstrak

Telekomunikasi, jaringan dan internet telah membentuk sebuah sistem terintegrasi yang mampu mengolah, menyimpan, dan mendistribusikan informasi dalam bentuk konten. Salah satu metode yang banyak dipakai adalah aplikasi berbasis web yang akan mendistribusikan konten agar bisa diakses melalui peramban web. Dalam sebuah website dengan struktur lengkap terdapat beberapa komponen yang menjadi bagian utama sehingga sebuah halaman web bisa dijalankan, antara lain server, jaringan, dan client.

Pemrograman web merupakan salah satu mata kuliah penting dalam mengiringi dan menjawab tantangan perkembangan teknologi website. Dalam kuliah pemrograman web, mahasiswa dituntut untuk dapat mengimplementasikan ilmu yang dipelajari menjadi sebuah aplikasi website yang dapat diakses oleh pengguna lainnya. Dalam penerapannya seringkali mahasiswa dihadapkan pada masalah ketersediaan alat, sumberdaya, dan kompatibilitas perangkat.

Pada Tugas Akhir ini akan dibangun sebuah sistem untuk mengakomodasi implementasi pemrograman web secara live tanpa melakukan instalasi perangkat memanfaatkan peramban

dengan teknologi peer-to-peer pada WebRTC.

Hasil uji coba terhadap sistem live coding secara peer-to-peer menunjukkan bahwa tidak ada masalah kompatibilitas pada perangkat dengan arsitektur prosesor yang berbeda asalkan terinstal peramban Google Chrome versi 40.0 keatas. Namun, secara performa menunjukkan bahwa kecepatan muat website dengan teknologi peer-to-peer lebih lambat.

***Kata-Kunci**: Javascript, WebRTC, NodeJS, Peer-to-Peer.*

DESIGN AND IMPLEMENTATION OF LIVE CODING ACTIVITY APPLICATION IN WEB PROGRAMMING USING PEER-TO-PEER IN BROWSER BASED ON WEBRTC

Name	:	**MUHAMAD ROHMAN**
NRP	:	**5111100043**
Major	:	**Informatics FTIf**
Supervisor I	:	**Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D**
Supervisor II	:	**Ir. Muchammad Husni, M.Kom.**

Abstract

Telecommunication, network, and internet has successfully formed an integrated system which is able to proceed, save, and distribute information in the form of content. Web based application is one of the methods utilized to distribute the content in order to make it easy accessed by web. In a complete structured website, there are some important components namely server, network, and client which help in running the web.

Web programming is one of the essential subject in trailing and responding the challenge of web technology development. In this subject, students are expected to implement the knowledge they have got into a web application that can be used by other users. In the implementation, it is often found that students have problems with tools or sources availability and device compatibility.

This final report built a system to accommodate live web programming implementation without installing the device by engaging to peer-to-peer technology by WebRTC.

The result showed that from the peer-to-peer coding there is no compatibility problem in a device that has different architect

processor unless it has been installed with Google Chrome 40.0 version and above. However, the performance shown that website loading speed of peer-to-peer technology is slower.

***Keywords**: Javascript, WebRTC, NodeJS, Peer-to-Peer.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE SUMBER

BAB 1

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Telekomunikasi dan internet telah membentuk sebuah sistem yang semakin terintegrasi untuk mengolah, menyimpan, mengakses, dan mendistribusikan informasi dan mengelola konten. Salah satu bagian penting dari internet adalah website, yang akan mendistribusikan dan mengelola konten agar bisa diakses melalui peramban.

Web *server* merupakan salah satu elemen penting dalam pengembangan sebuah sistem berbasis web, dengan adanya web *server* semua konten dapat didistribusikan secara luas melalui jaringan internet. Dalam perkembangannya, sebuah website membutuhkan peramban untuk mengolah data dan menjalankan aplikasi web agar informasi yang disampaikan dapat dibaca dan dimengerti oleh orang awam sekalipun. Oleh karena itu, peramban menjadi salah satu bagian yang berperan penting dalam mengiringi perkembangan teknologi website.

Dalam mata kuliah pemrograman web, mahasiswa harus melakukan instalasi aplikasi *server* seperti NodeJS untuk menunjang proses pembelajaran. Hal ini tentu kurang praktis dan efisien karena memakan sumberdaya seperti ruang penyimpanan. Selain *server* lokal pada setiap perangkat yang dimiliki oleh mahasiswa, metode alternatif yang bisa digunakan adalah pemanfaatan *server* manajemen konten terpusat yang bisa dimanfaatkan oleh banyak pengguna layaknya sebuah *web hosting*, namun memiliki kelemahan bila diakses oleh banyak pengguna secara bersamaan akan memberatkan kinerja *server*.

Dengan memanfaatkan Javascript yang memiliki kemampuan dijalankan pada sisi *client*, sebuah peramban dapat dimanfaatkan untuk memproses aplikasi layaknya sebuah *server*. Memanfaatkan kemampuan koneksi *peer-to-peer* pada WebRTC maka dapat diciptakan sebuah mekanisme satu jaringan utuh dengan memanfaatkan beberapa peramban tanpa melakukan instalasi *plugin* apapun.

Desentralisasi jaringan *peer-to-peer* memiliki keuntungan yang lebih dibandingkan dengan jaringan *client-server* tradisional. Jaringan *peer-to-peer* menyeimbangkan diri secara otomatis tanpa menambah waktu pencarian alamat tujuan dan tanpa harus menggunakan suatu sumber-sumber terpusat. Peer-to-peer memanfaatkan mesin –mesin perangkat yang digunakan *end user* yang terasosiasi dalam satu jaringan . Setiap penambahan *node* baru pada jaringan, menambah potensi sumberdaya pemrosesan yang lebih kuat dan *bandwith* yang lebih besar untuk jaringan tersebut. Ditambah lagi, karena sumber - sumbernya terdesentralisasi, generasi kedua (2G) dari jaringan *peer-to-peer* telah berhasil secara virtual mengeliminasi seluruh biaya yang berhubungan dengan infrastruktur terpusat yang besar.

Muncullah gagasan lain mengenai konsep *client* yang dapat difungsikan layaknya sebagai *server* memanfaatkan peramban sebagai media operasionalnya, menggunakan teknologi *peer-to-peer* sebagai media komunikasi sehingga beban kerja akan diarahkan langsung ke masing - masing *node* memanfaatkan WebRTC agar konsep *peer-to-peer* dapat dijalankan pada peramban.

Oleh karena itu, pada Tugas Akhir ini diusulkan pembuatan sistem manajemen konten web yang berjalan pada peramban untuk menangani dan menyediakan monitoring pengerjaan *live coding*[1] sehingga dapat dilakukan monitoring hasil pengerjaan kode program secara langsung melalui akses URL tanpa harus

menyediakan *server* fisik, basis data dan *dashboard* untuk setiap pengguna. Semua kebutuhan seperti *server* dan basis data dijalankan pada masing – masing peramban menggunakan konsep *peer-to-peer* pada WebRTC sehingga tidak melibatkan *server* pusat dalam proses pertukaran data untuk mengurangi beban *server* utama. Selain itu, diharapkan dalam memonitoring *live coding*, semua pengguna memiliki satu *platform* yang sama dan terpusat.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana membuat layanan yang memfungsikan peramban web sebagai penyedia konten sementara untuk aktivitas *live coding* secara *peer-to-peer*?
2. Bagaimana membuat *server* yang dapat digunakan untuk mengatur distribusi konten antar peramban web?
3. Bagaimana membuat antarmuka *editor* teks untuk melakukan aktivitas pemrograman secara langsung pada peramban web?
4. Bagaimana performa penggunaan teknologi *peer-to-peer* pada kasus penggunaan *editor* teks untuk pemrograman web?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Aplikasi berbasis web dengan bahasa pemrograman Javascript menggunakan kerangka kerja Express dan *server engine* NodeJS.

2. Sistem hanya dapat menerima dan menjalankan konten HTML, Javascript, CSS, dan file gambar (.jpg, .png, .jpeg)
3. Uji coba dilakukan pada peramban yang mendukung sistem layanan WebRTC yaitu Google Chrome 40.0 (atau versi yang lebih baru).

1.4 Tujuan

Tujuan dari dibuatnya Tugas Akhir ini adalah menghasilkan sistem dan perangkat lunak manajemen konten web agar memungkinkan peramban web dapat difungsikan sebagai *client-server* penyedia konten sementara untuk ujicoba baris kode dan dapat diakses oleh pengguna lain secara *peer-to-peer*.

1.5 Manfaat

Manfaat yang diberikan dengan pembuatan Tugas Akhir ini adalah memberikan kemudahan kepada pengguna untuk membuat *server* manajemen konten web sementara dengan memanfaatkan peramban untuk memudahkan implementasi pembelajaran pemrograman web sederhana tanpa harus melakukan instalasi aplikasi *editor* teks dan *server*.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

- **Penyusunan Proposal Tugas Akhir**
  Penyusunan proposal Tugas Akhir dilaksanakan untuk merumuskan masalah serta melakukan penetapan desain dasar sistem yang akan dikembangkan dalam pelaksanaan Tugas Akhir ini.

- **Studi Literatur**
  Untuk membantu proses pengerjaan Tugas Akhir, diperlukan studi lebih lanjut mengenai penggunaan komponen-komponen terkait dengan sistem yang akan dibangun.
- **Desain dan Perancangan**
  Dalam rangka memerinci lebih jauh mengenai bagaimana memanfaatkan komponen-komponen sistem untuk membangun sistem secara utuh, diperlukan proses desain dan perancangan dari sistem. Hasil analisis dan desain kemudian ditetapkan menjadi rancangan dasar implementasi sistem.
- **Implementasi Sistem**
  Hasil analisis dan desain kemudian diimplementasikan melalui komponen-komponen perangkat lunak pendukung.
- **Uji Coba dan Evaluasi**
  Untuk mengukur kemampuan sistem untuk menangani pengguna dari segi fungsionalitas dan performa, dilakukan proses uji coba dan evaluasi untuk mengetahui sejauh mana sistem dapat berjalan sesuai dengan yang diharapkan.

1.7 Sistematika Laporan

Buku Tugas Akhir ini terdiri dari beberapa bab yang dijelaskan sebagai berikut:

- **Bab 1. Pendahuluan**. Bab ini berisikan latar belakang, rumusan masalah, tujuan, manfaat, metodologi dan sistematika penulisan yang digunakan sebagai dasar penyusunan Tugas Akhir.
- **Bab 2. Tinjauan Pustaka**. Bab ini berisikan teori penunjang yang berhubungan dengan Tugas Akhir.
- **Bab 3. Desain dan Perancangan**. Bab ini membahas

desain dasar dari sistem dan perangkat lunak yang akan dirancang sebagai bagian dari pengerjaan Tugas Akhir.

- **Bab 4. Implementasi**. Bab ini membahas hasil implementasi dari rancangan sistem beserta tekniknya.
- **Bab 5. Pengujian dan Evaluasi**. Bab ini membahas mengenai teknik uji coba dan hasil keluarannya sebagai bahan evaluasi terhadap hasil Tugas Akhir.
- **Bab 6. Penutup**. Bab ini berisi kesimpulan dari hasil uji coba serta saran untuk pengembangan Tugas Akhir selanjutnya.

BAB 2

TINJAUAN PUSTAKA

Pada bab ini berisi penjelasan teori-teori yang berkaitan dengan pengimplementasian perangkat lunak.Bab ini bertujuan untuk memberikan gambaran secara umum mengenai teori serta alat bantu yang digunakan pada implementasi perangkat lunak pada Tugas Akhir ini.

2.1 WebRTC

WebRTC *(Web Real Time Communication)* adalah sebuah standar baru yang memungkinkan sebuah peramban berkomunikasi secara *real time* dengan menggunakan arsitektur *peer-to-peer*. Hal ini berkonsen pada koneksi audio/video(dan data) antar peramban web. Hal ini merupakan evolusi di dunia teknologi web, karena untuk pertama kalinya, pengembang web mampu membangun aplikasi *real time* tanpa *plug-in*. WebRTC membutuhkan STUN *(Session Traversal Utilities for NAT) server* agar pengguna aplikasi dapat saling terhubung. TURN *(Traversal Using Relay around NAT) server* merupakan fitur dari STUN *server* yang menempatkan TURN *server* seolah-olah berada ditengah-tengah antara kedua pengguna aplikasi yang mengurusi pengiriman data [2].

API *(Application Programming Interface)* adalah sekumpulan perintah, fungsi, dan protokol yang dapat digunakan oleh pengembang aplikasi saat membangun sistem perangkat lunak[3]. API memungkinkan pengembang web menggunakan fungsi standar untuk berinteraksi dengan sistem penyedia API. API dapat menjelaskan cara sebuah tugas *(task)* dilakukan. Dalam pemrograman prosedural seperti bahasa C, aksi biasanya dilakukan dengan media pemanggilan fungsi. Oleh karena itu, API menyertakan penjelasan dari fungsi yang disediakan. WebRTC API berupa fungsi Javascript yang dapat digunakan untuk berinteraksi dengan aplikasi peramban web.

2.2 NodeJS

NodeJS adalah *open source, cross-platform runtime environtment* pada sisi *server* dan aplikasi jaringan. NodeJS merupakan aplikasi yang ditulis dalam Javascript, dan dapat dijalankan pada OS X, Microsoft Windows, Linux, FreeBSD, NonStop, dan IBM i.

Merupakan sebuah platform yang dibangun di atas Chrome's Javascript *runtime* dengan teknologi V8 yang mendukung proses *server* yang bersifat *long-running*. Tidak seperti *platform* moderen yang mengandalkan *multithreading*, NodeJS memilih menggunakan asynchronous I/O eventing. Karena inilah NodeJS mampu bekerja dengan konsumsi memori rendah [4][5].

Teknologi yang tidak memanfaatkan *multi-thread* ini memudahkan pengembang yang terkadang kesulitan mengatur sumberdaya yang menggunakan thread. Karena tidak mungkin ada sumberdaya yang terkunci karena *thread* yang berjalan. Akhirnya banyak yang memanfaatkan kemampuan dasar NodeJS sebagai web *server*.

Kode Sumber 2.1: Contoh Penggunaan NodeJS sebagai Web Server

```
1  var http =  require('http');
2  const PORT=8080;
3
4  function handleRequest(request, response)
       {
5   response.end('It Works !! Path Hit: '+
       request.url);
6  }
7
8  var server = http.createServer(
     handleRequest);
9
10 server.listen(PORT, function(){
```

```
11      console.log("Server is running on port:
            %s", PORT)});
```

Dengan adanya *callback* untuk setiap penggunaan fungsi, memungkinkan setiap pemanggilan fungsi yang tidak menghasilkan apapun, NodeJS akan menuju mode *sleep*. Kode Sumber 2.1 menunjukkan contoh penggunaan NodeJS untuk http *server*.

2.3 Data Channel

Data channel didefinisikan sebagai API pada tingkatan aplikasi yang bisa mencerminkan API untuk WebSocket, yang menjelaskan *stream* dua arah dari data dan bidang tekstual yang disebut ”label” dan digunakan untuk mengidentifikasi arti dari data channel[6]. Realisasi dari sebuah data channel merupakan pasangan antara satu *incoming stream* dengan *outgoing* SCTP *(Stream Control Transmission Protocol) stream* yang memiliki penanda SCTP *stream* yang sama. Penanda SCTP *stream* dipilih tergantung pada protokol dan implementasinya. Hal ini memungkinkan terjadinya komunikasi dua arah.

2.4 Peer

Peer adalah modul dari NodeJS yang dibangun untuk membantu koneksi *client* peer.js pada aplikasi website *peer-to-peer*. Peer digunakan pada *session metadata* dan *candidate signaling*. Ada dua koneksi yang bisa dibuat oleh modul *peer* tersebut yaitu koneksi data dan koneksi media seperti audio dan video[7].

Sebelum koneksi dilakukan, modul *peer* harus dijalankan terlebih dahulu pada *server*. Sebuah koneksi membutuhkan identitas seperti IP pada komputer. Ketika *peer* ingin berhubungan, *peer* membutuhkan identitas *peer* yang dituju.

Peer akan menerima *event* `Connection` untuk koneksi dan atau `Call` untuk koneksi media ketika *peer* menerima permintaan untuk saling berhubungan. Data dari masing - masing *peer* dikirim melalui web-socket.

2.5 Backbone JS

Ketika bekerja dalam lingkungan aplikasi yang menjalankan banyak Javascript, salah satu hal penting yang harus diperhatikan adalah bagaimana menghentikan ikatan data kedalam DOM *(Document Object Model)*. Dengan Backbone JS, semua data akan direpresentasikan sebagai model yang bisa dibuat, divalidasi, dihapus, dan disimpan kedalam *server*.

Secara harfiah, Backbone merupakan upaya untuk mengatur struktur (model dan *collection*) dan antarmuka pengguna (*view* dan URL) secara primitif yang mana akan berguna dalam pembuatan aplikasi berbasis web dengan Javascript[8].

Backbone JS memberikan struktur untuk aplikasi web dengan menyediakan model dengan *key-value binding* dan *custom event*, *collection* dengan API untuk fungsi - fungsi *enumerable*, *view* dengan *event handling* deklaratif, dan menghubungkan itu semua menjadi API melalui antarmuka RESTful JSON.

2.6 Handlebar JS

Handlebar JS merupakan modul Javascript yang berfungsi sebagai prosesor templat yang dinamis yang akan menghasilkan halaman HTML untuk menghemat waktu dari penyusunan kode program secara manual dan berulang - ulang[9]. Memanfaatkan Javascript, Handlebar JS mengkomunikasikan file JSON dengan HTML untuk pemrosesan templat.

Gambar 2.1: Contoh Penggunaan Handlebar JS untuk Templat

2.7 MongoDB

Merupakan salah satu NoSQL (Not only SQL) terkenal yang dirancang untuk mengelola polimorfik, obyek, dan struktur data yang terus berkembang. MongoDB adalah basis data *open-source* yang memungkinan mengubah skema dengan cepat sementara fungsi yang diharapkan dari basis data tradisional masih berjalan [10][11].

Gambar 2.2: Contoh Bentuk Penyimpanan Data MongoDB

Model penyimpanan yang menyerupai JSON membuat pengguna dapat dengan mudah mengakses dan mengubah data yang ada. Karena penyimpanannya yang menyerupai JSON, membuat struktur penyimpanan dapat berubah sewaktu-waktu tanpa mengubah konfigurasi sebelumnya.

2.8 Javascript

Javascript merupakan bahasa pemrograman yang umum digunakan dalam pemrograman web. Bahasa pemrograman ini didukung pada berbagai peramban, salah satunya Google Chrome. Karena sudah umum digunakan, banyak peramban moderen berbasis *desktop* maupun *mobile* saat ini menanamkan dukungan Javascript [5].

Javascript merupakan bahasa pemrograman yang berjalan pada sisi *client*. Hal ini menyebabkan setiap eksekusi perintah dilakukan oleh peramban dimana pengguna mengakses situs web. Penggunaan Javascript sendiri berdampingan dengan HTML dan CSS, dimana Javascript dapat digunakan untuk memanipulasi konten dan desain dari sebuah situs web.

2.9 Asyncronous Javascript and XML (AJAX)

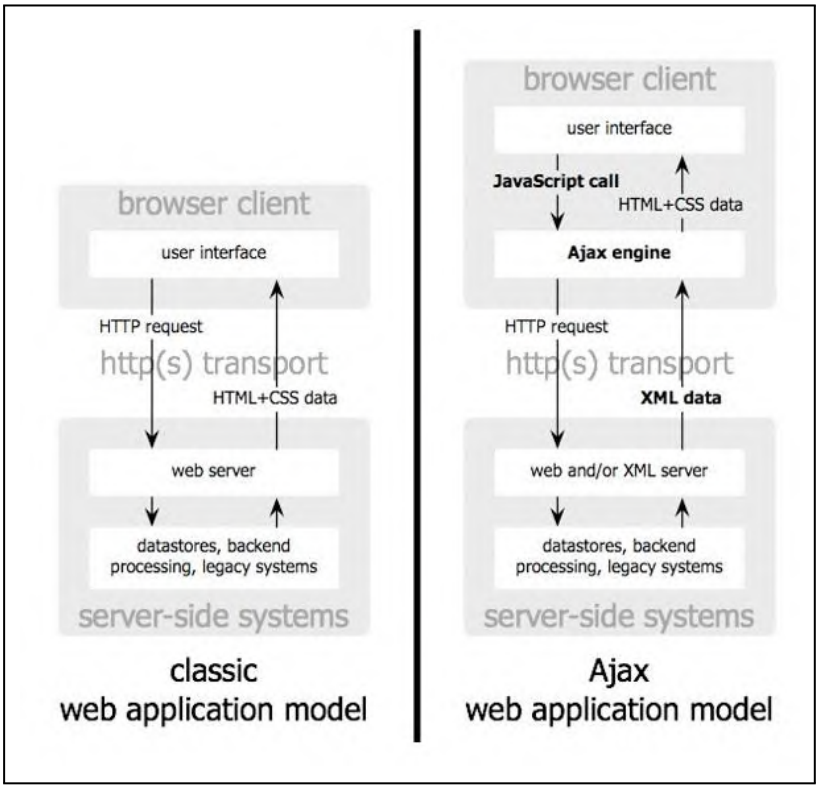
AJAX merupakan salah satu teknik pengembangan halaman web yang memungkinkan sebuah halaman situs dapat menerima konten secara dinamis melalui *server* [12]. Pertukaran data antara *client* dengan *server* dilakukan secara *asynchronous*, sehingga tidak mengganggu tampilan situs. Implementasi AJAX sendiri memanfaatkan gabungan beberapa komponen, yaitu:

- Situs dengan XHTML dan CSS
- Tampilan dinamis dan interaktif dengan memanfaatkan *Document Object Model* (DOM).
- Pertukaran dan manipulasi data menggunakan XML dan XSLT.
- Menerima data secara *asynchronous* menggunakan XMLHttpRequest.
- Javascript untuk menggabungkan semua komponen tersebut.

Gambar 2.3 menunjukan bagaimana perbandingan antara

aplikasi web klasik tanpa AJAX dengan aplikasi web menggunakan AJAX. Aplikasi web tanpa AJAX melakukan permintaan ke web *server* kemudian memberikan respon dalam bentuk halaman HTML, sehingga pengguna akan mengalami perpindahan halaman pada peramban. Berbeda dengan situs yang mengimplementasikan AJAX, permintaan ke web *server* dilakukan melalui perantara Javascript dengan memanggil obyek XMLHttpRequest. Web *server* akan mengirimkan respon berupa data XML yang kemudian diolah oleh peramban untuk kemudian ditampilkan kedalam bentuk HTML dan CSS.

Meskipun obyek yang digunakan adalah XMLHttpRequest, namun pada perkembangannya respon yang diolah tidak selalu berupa XML. Saat ini, jenis respon yang populer digunakan adalah format JSON *(Javascript Object Notation)* yang umumnya digunakan bersama dengan jQuery. Selaind alam bentuk teks yang telah terformat, respon juga dapat berupa *plain text* yang dibuat dalam format sesuai kebutuhan *programmer*.



Gambar 2.3: Perbandingan Antara Aplikasi Web Klasik denganAplikasi Web Berbasis AJAX

BAB 3

DESAIN DAN PERANCANGAN

Desain dan perancangan merupakan bagian awal dalam pengembangan perangkat lunak, berisi perencanaan teknis dari aplikasi yang akan dibuat, pada bab ini akan dijelaskan metode-metode yang digunakan dalam pengerjaan Tugas Akhir ini.

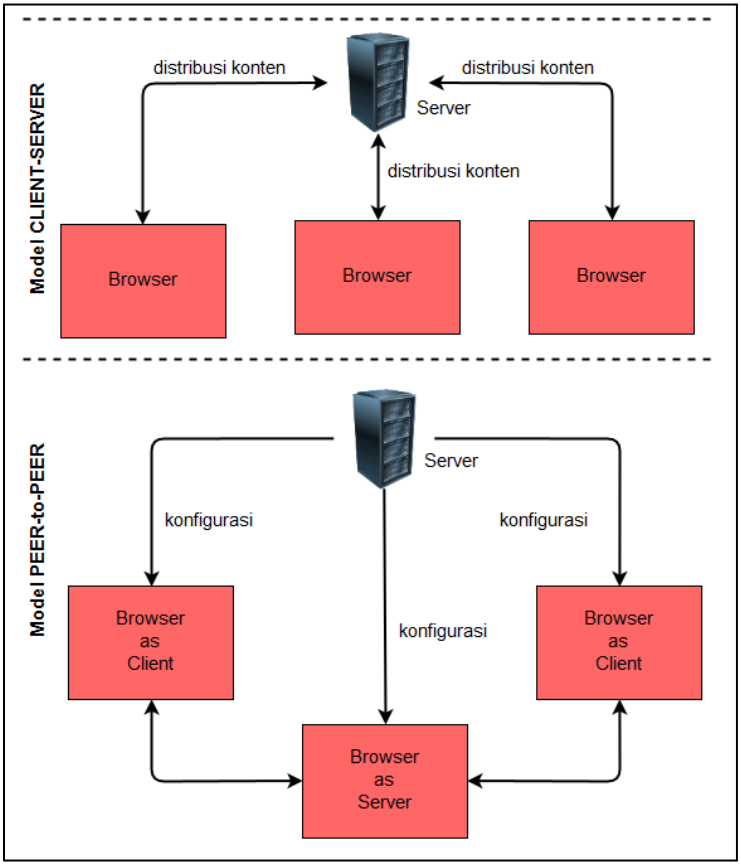
3.1 Deskripsi Umum

Pada Tugas Akhir ini dibangun sebuah sistem untuk melayani distribusi konten pada pemrograman web secara *peer-to-peer* dengan tujuan untuk mempermudah implementasi pembelajaran pemrograman web sederhana menggunakan peramban yang ada. Sistem akan menyediakan dasbor yang difungsikan untuk melakukan aktifitas pemrograman web seperti membuat kode CSS untuk kebutuhan tampilan, pengaturan rute halaman, penyimpanan data menggunakan basis data, dan penyajian konten menggunakan kode HTML.

Aplikasi Tugas Akhir ini dibangun pada platform NodeJS memanfaatkan *framework* Express. Memanfaatkan WebRTC, aplikasi akan berjalan melalui mekanisme *peer-to-peer* untuk proses komunikasi antar peramban yang akan bertindak sebagai penyedia layanan *(server)* maupun pengakses layanan *(client)*. Pengguna dapat melakukan inisiasi peramban yang dimilikinya untuk difungsikan sebagai *server* bagi peramban lain yang akan menerima distribusi konten dengan melakukan permintaan kepada *server* utama. Server utama akan mengatur inisiasi berupa penentuan ID peer dan *routing*, bila koneksi antar *node* sudah terbentuk, selanjutnya mekanisme *peer-to-peer* akan ditangani oleh Peer JS melalui TURN *server* (fitur dari STUN) yang seolah-olah berada di tengah-tengah sebagai penghubung antar peer.

Sistem ini menggantikan model *client-server* pada umumnya

yang menggunakan protokol HTTP untuk tujuan komunikasi pertukaran data, dengan protokol *peer-to-peer* dan menjadikan peramban dapat berfungsi layaknya *server* bagi peramban lain yang berperan sebagai *client*, perbedaan metode ini dijelaskan pada Gambar 3.1.



Gambar 3.1: Perbandingan Antara Model Client-Server dengan Peer-to-Peer

Melalui sistem ini pengguna dapat melakukan aktivitas *live coding* tanpa harus melakukan instalasi IDE *(Integrated Development Environtment)*, *server*, dan basis data. Selain itu, *server* utama dapat menampung pengguna yang tak terbatas karena setelah inisiasi peer, semua proses penyimpanan dan pengolahan data akan dibebankan ke masing-masing *client*.

3.1.1 Pemrograman Web Secara Peer-to-Peer

Pemrograman secara *peer-to-peer* mengacu pada kompilasi kode program aplikasi web pada *client* melalui sebuah jaringan *peer-to-peer*. Terdapat dua *node* berupa peramban yang memiliki peran sebagai *server* dan *client*. Pengaplikasisn kode sumber dilakukan pada peramban yang berperan sebagai *server*, kode program dikirimkan melalui jaringan *peer-to-peer* ke peramban yang bertindak sebagai *client* untuk kemudian dilakukan kompilasi, makna dari *live coding* disini adalah sifat peramban yang berperan sebagai *server* hanya menyediakan ketersediaan akses secara *temporary* sehingga bila peramban yang berperan sebagai *server* dimatikan, maka semua hasil pemrograman web akan hilang.

Hal ini sejalan dengan tujuan pembuatan aplikasi *live coding* untuk pengaplikasian contoh kode program pada pembelajaran web, sehingga dapat diketahui bagaimana sebuah potongan kode akan terlihat ketika dijalankan.

Pemanfaatan arsitektur *peer-to-peer* mengacu pada kebutuhan aplikasi yang harus dijalankan pada peramban dan komunikasi data antar peramban yang dapat dilakukan saat ini adalah komunikasi menggunakan WebRTC.

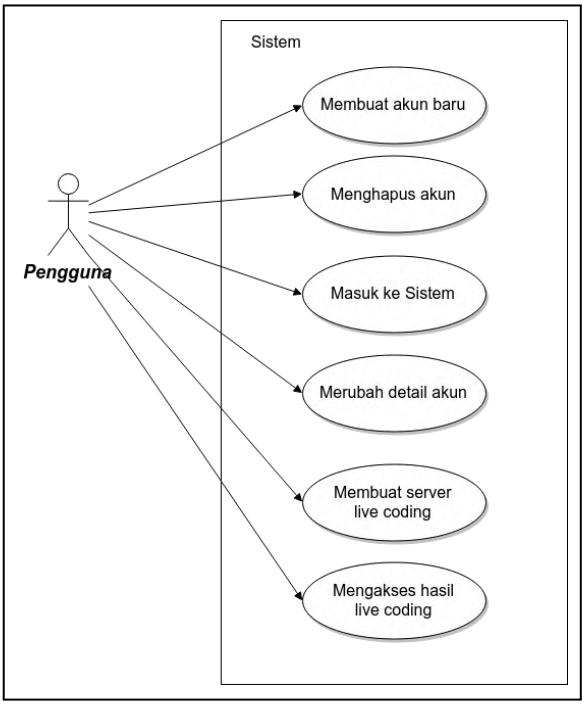
Live coding mengacu pada pengaplikasian kode secara *live* dan tidak disediakan penyimpanan data permanen, karena peramban tidak memiliki media penyimpanan permanen.

3.2 Kasus Penggunaan

Gambar 3.2 menampilkan diagram kasus penggunaan sistem secara umum dengan penjelasan pada Tabel 3.1. Diagram kasus penggunaan menggambarkan fungsionalitas sistem beserta aktor yang terlibat di dalamnya, dalam Tugas Akhir ini hanya terdapat satu aktor yaitu Pengguna.

Tabel 3.1: Penjelasan Diagram Kasus Penggunaan

No	Nama	Aktor	Deskripsi
UC01	Membuat akun	Pengguna	Pengguna dapat membuat akun baru
UC02	Masuk ke sistem	Pengguna	Pengguna dapat masuk ke sistem menggunakan detail akun pendafataran
UC03	Menghapus akun	Pengguna	Pengguna dapat menghapus akun yang dibuat sebelumnya
UC04	Mengubah detail akun	Pengguna	Pengguna dapat mengubah detail akun
UC05	Membuat *server*	Pengguna	Pengguna dapat membuat *server* baru
UC06	Mengakses hasil *live coding*	Pengguna	Pengguna dapat mengakses hasil *live coding*



Gambar 3.2: Diagram Kasus Penggunaan Sistem

3.3 Arsitektur Sistem

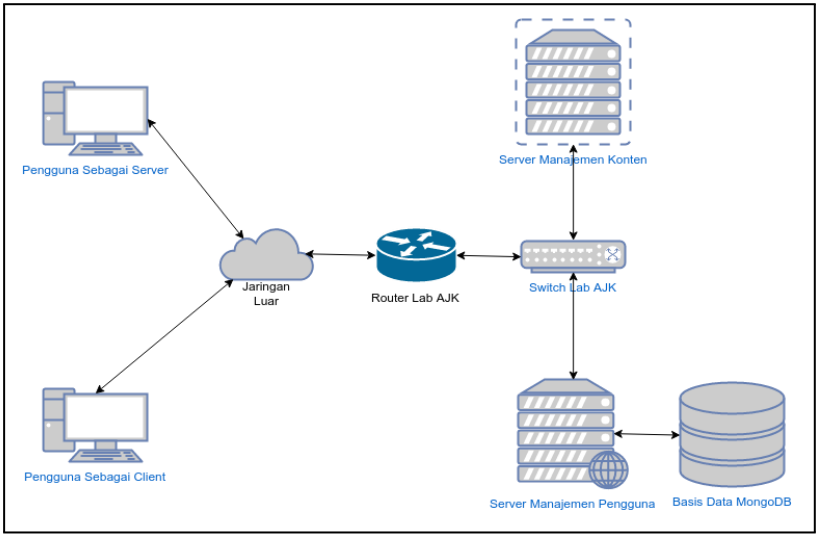
Pada sub-bab ini, dibahas mengenai tahap analisis dan kebutuhan bisnis dan desain dari sistem yang akan dibangun.

3.3.0.1 Desain Umum Sistem

Gambar 3.3 menunjukan desain umum sistem, terdiri dari *server* manajemen pengguna yang akan menangani pencatatan akun pengguna, *server* manajemen konten yang akan menangani pengaturan koneksi ke STUN *server* untuk arsitektur *peer-to-peer* dan pengelolaan konfigurasi konten untuk

memanajemen baris kode. Baik *server* manajemen konten maupun *server* manajemen pengguna dibangun diatas teknologi NodeJS.

Untuk menjalankan fungsionalitasnya, diperlukan dua buah perangkat komputer yang terinstal peramban Google Chrome versi 40 keatas, yang akan bertindak sebagai *Browser as Server* dan *Browser as Client*. *Browser as Server* akan bertindak sebagai tempat untuk mengimplementasikan baris kode, basis data, dan pengunggahan gambar, yang selanjutnya akan diakses oleh *Browser as Client* sebagai pengakses dan penguji keberhasilan kompilasi baris kode yang di implementasikan melalui arsitektur *peer-to-peer* memanfaatkan WebRTC, selanjutnya mekanisme ini disebut sebagai mekanisme *live coding*.



Gambar 3.3: Desain Sistem Secara Umum

3.3.0.2 Desain Manajemen Pengguna

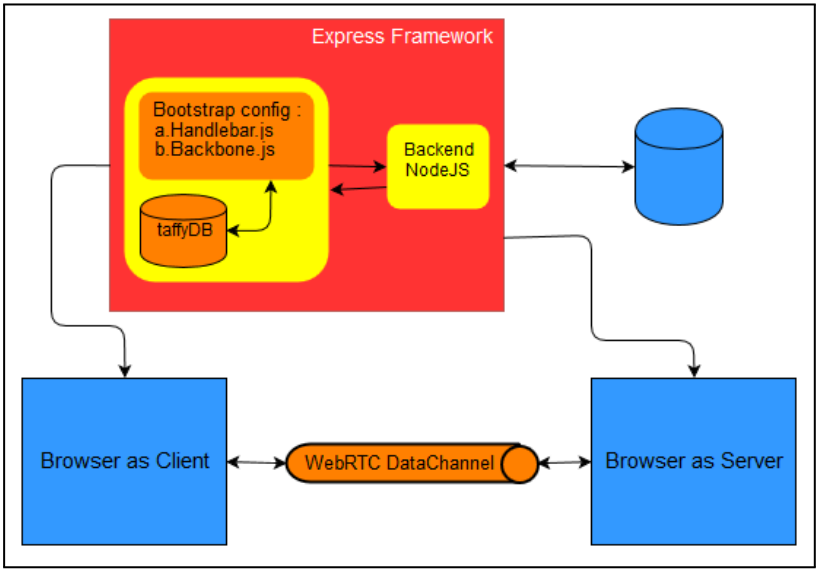
Manajemen pengguna dibuat dengan kerangka kerja dari bahasa Javascript yaitu Express. Manajemen pengguna akan menangani segala bentuk operasi yang berkaitan dengan pengguna dan basis data. Tabel 3.2 menunjukan daftar rute yang diakomodasi oleh aplikasi beserta aksinya.

Tabel 3.2: Daftar Rute Aplikasi Manajemen Pengguna

No	Rute	Metode	Hak akses	Aksi
1	`/`	GET	Tidak ada	Menampilkan halaman utama aplikasi
2	`/login`	GET	Tidak ada	Menampilkan halaman masuk aplikasi
3	`/login`	POST	Tidak ada	Melakukan autentikasi *email* dan *password*
4	`/signup`	GET	Tidak ada	Menampilkan halaman pendaftaran
5	`/signup`	POST	Tidak ada	Menambahkan detail pendaftaran kedalam basis data
6	`/account`	GET	Pengguna	Menampilkan profil akun pengguna
7	`/account/profile`	POST	Pengguna	Melakukan update informasi pengguna

Tabel 3.2: Daftar Rute Aplikasi Manajemen Pengguna

8	`/account/password`	POST	Pengguna	Melakukan update informasi *password* pengguna
9	`/account/delete`	POST	Pengguna	Menghapus akun pengguna dari basis data



Gambar 3.4: Diagram Arsitektur Sistem

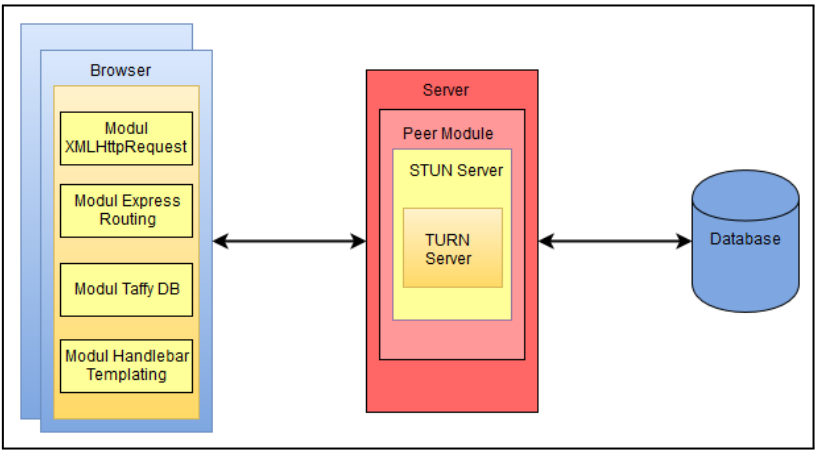
3.3.0.3 Desain Manajemen Konten

Arsitektur sistem pada aplikasi ini seperti Gambar 3.4, aplikasi ini berjalan pada sebuah *server* yang berperan sebagai *server* utama dan bisa diakses oleh peramban yang akan berperan sebagai *Browser as Server* maupun *Browser as Client* yang

dibuat berdasarkan data yang terdaftar pada basis data MongoDB.

Komunikasi antara peramban dengan fitur STUN *server* untuk koneksi *peer-to-peer* antar peramban ini dijelaskan pada Gambar 3.5, setiap modul bekerja untuk memenuhi kebutuhan sistem. pada peramban terdapat modul - modul yang digunakan untuk memproses data yang nantinya diterima oleh peramban, antara lain:

1. Modul `XMLHttpRequest` digunakan untuk menangani `HttpRequest` menggunakan AJAX sehingga tidak perlu melakukan pemuatan ulang pada peramban.
2. Modul `Express Routing` digunakan untuk mengatur *routing* alamat `URI`
3. Modul `TaffyDB` digunakan untuk mengatur basis data file aplikasi.
4. Modul `Handlebar` digunakan untuk mengatur templat aplikasi.



Gambar 3.5: Arsitektur Sistem Aplikasi

Gambar 3.5 merupakan gambaran arsitektur dari

pembangunan aplikasi. Aplikasi terdiri dari dua modul utama. Modul pertama adalah peramban yang berfungsi sebagai antarmuka pengguna. Pada sisi pengguna terdapat Peer JS *client* untuk membantu koneksi peer ke *server* serta menghubungkan satu peer dengan peer lainnya. Pengolah data web juga berada di *client*. Setelah koneksi terjalin dan terjadi pertukaran data, maka data mentah dapat dieksekusi di *client*.

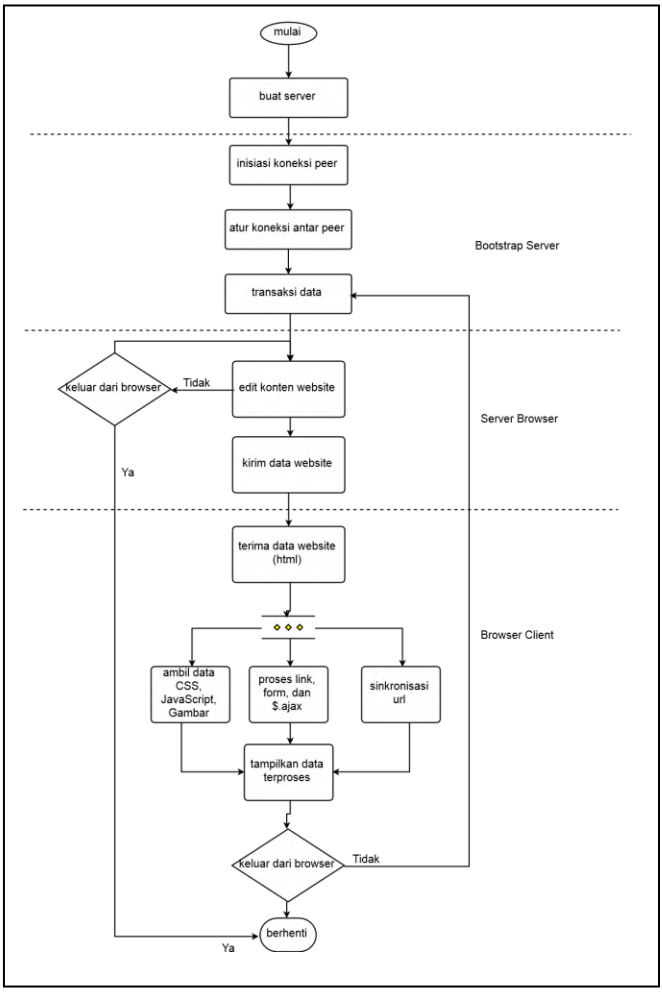
Modul selanjutnya adalah *server* yang terdiri atas NodeJS yang berfungsi untuk menjalankan modul peer untuk membantu koneksi PeerJS *client* pada aplikasi website *peer-to-peer*. Di dalam modul peer terdapat STUN *server* dan TURN *server*. TURN *server* merupakan fitur dari STUN *server* yang berfungsi menghubungkan peer satu dengan yang lainnya seolah-olah TURN *server* berada ditengah-tengah kedua peer. Basis data yang digunakan adalah MongoDB, digunakan untuk menyimpan identitas dari setiap pengguna. Identitas tersebut digunakan untuk mengenali aplikasi yang dibuat oleh pengguna. Setiap pengguna akan dibuatkan identitas unik, dan aplikasi tidak dapat menerima identitas yang sama.

Agar dapat menjalankan fungsinya, maka alur kerja dari kesatuan aplikasi ini dirancang seperti pada Gambar 3.6 dengan penjabaran sebagai berikut:

1. Pengguna melakukan inisialisasi koneksi peer sebagai *Browser as Server* ke *server* manajemen konten.
2. Server manajemen konten menerima semua permintaan koneksi dan memberikan identitas untuk peer yang terkoneksi.
3. *Server* manajemen konten mengirimkan file konfigurasi untuk *Browser as Server*
4. Konfigurasi dimuat oleh *Browser as Server* dalam bentuk antarmuka *editor* teks. Pengguna dapat mengaplikasikan baris kode di dalamnya.
5. Semua baris kode di *Browser as Server* disimpan dalam

local storage peramban yang digunakan.

6. Peramban baru melakukan permintaan koneksi ke *server* manajemen konten sebagai *Browser as Client* menyertakan identitas *Browser as Server* tujuan.
7. *Server* manajemen konten mengirimkan file konfigurasi untuk *Browser as Client* dan identitas peer untuk saling berkomunikasi antara *Browser as Server* dan *Browser as Server*
8. Setelah komunikasi *peer-to-peer* terjalin, *Browser as Client* dapat melakukan permintaan ke *Browser as Server*, sehingga proses pengiriman data dapat saling dilakukan.



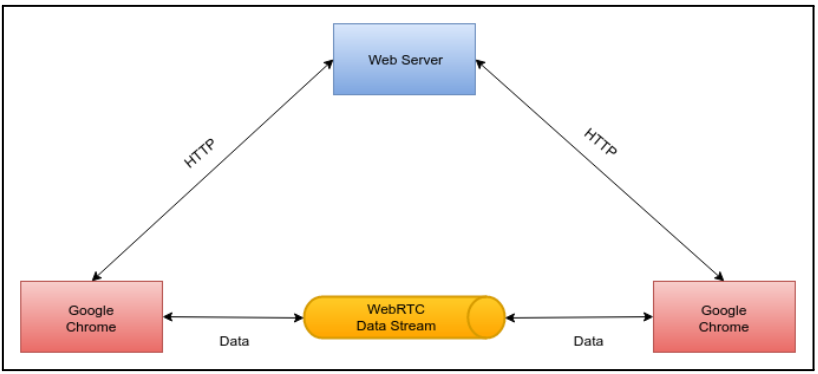
Gambar 3.6: Gambaran Umum Alur Kerja Aplikasi

3.4 Perancangan Diagram Alir Data Aplikasi Live Coding

Pada bagian ini akan dibahas mengenai gambaran aliran data dan fungsionalitas sistem secara umum. Hal ini direpresentasikan berupa diagram konteks. Pada sistem ini web peer melakukan

3.4.1 Perancangan Diagram Konteks Aplikasi Live Coding

Diagram konteks merupakan diagram alir yang menggambarkan sistem secara umum. Semua aktor eksternal serta aliran data masuk dan keluar sistem digambarkan dalam satu diagram, dimana keseluruhan sistem digambarkan dalam satu proses. Diagram konteks aplikasi ini ditunjukkan pada Gambar 3.7.



Gambar 3.7: Diagram Konteks Aplikasi

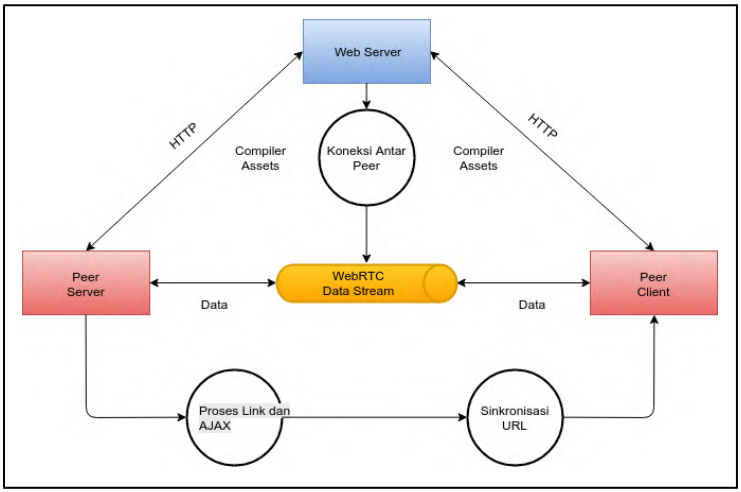
Seperti yang ditunjukan pada Gambar 3.7, sistem ini akan menerima HTTP *request* dari peramban untuk membuka aplikasi. Web *server* akan mengirim konten yang diminta oleh peramban. Setelah semua konten dimuat dengan sempurna, peramban akan meminta inisialisasi koneksi peer kepada *server*. Setelah inisialisasi dilakukan, koneksi peer kemudian

dibuat.Ketika peer melakukan komunikasi dengan peer lain, data dikirim melalui WebRTC `DataStream`

3.4.2 Diagram Alir Data Level 0 Aplikasi

Diagram alir data level 0 merupakan dekomposisi dari proses utama pada diagram konteks. Diagram ini menggambarkan fungsionalitas yang terjadi pada sistem ini. Diagram alir data level 0 ditunjukkan pada Gambar 3.8

Diagram pada Gambar 3.8 menunjukkan proses - proses yang terjadi pada sistem. Proses awal yang terjadi yaitu peramban meminta HTTP *request* kepada web *server* untuk membuka aplikasi. Web *server* mengirimkan semua konten yang dibutuhkan untuk proses kompilasi di *client*. Selanjutnya peramban meminta inisialisasi koneksi peer kepada *server*. *Server* melakukan inisialisasi koneksi kepada pengguna.



Gambar 3.8: Diagram Alir Data Level 0

Koneksi antar peer bisa dilakukan apabila peer mengetahui identitas peer lain. Koneksi yang terjadi antar peer dilakukan melalui WebRTC, sehingga konsep jaringan *peer-to-peer* yang utuh dapat dilakukan. Setelah koneksi dapat dilakukan, maka pertukaran data dapat dilakukan(*client* menerima data mentah *(raw data)* dari peer yang diminta)

3.5 Rancangan Antarmuka

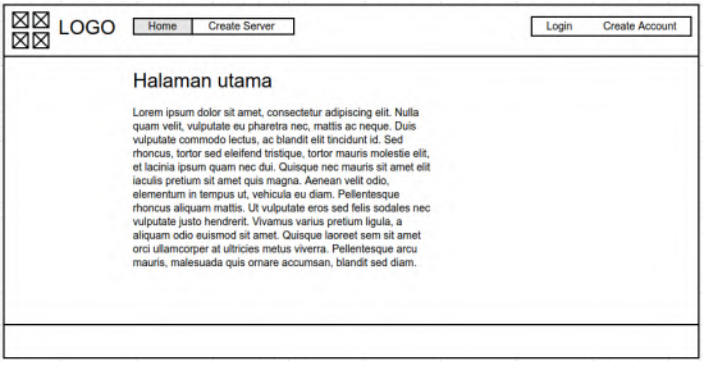
Antarmuka dibagi menjadi dua aplikasi yaitu aplikasi *server* manajemen pengguna dan aplikasi manajemen konten.

3.5.1 Rancangan Antarmuka Manajemen Pengguna

Merupakan antarmuka yang digunakan untuk aktivitas manajemen profil pengguna, seperti pembuatan akun baru, perubahan akun, dan menghapus akun.

3.5.1.1 Halaman Utama

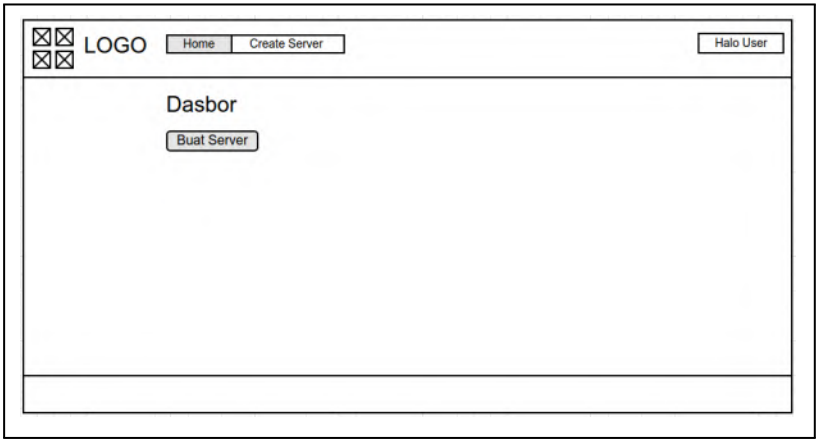
Seperti yang terlihat pada Gambar 3.9, halaman ini adalah halaman *default* ketika pertama kali mengakses ke sistem.



Gambar 3.9: Rancangan Antarmuka Halaman Utama

3.5.1.2 Halaman Dasbor Pengguna

Seperti yang terlihat pada Gambar 3.10, halaman ini adalah halaman pengguna membuat *server* peer yang selanjutnya disebut *Browser as Server*.

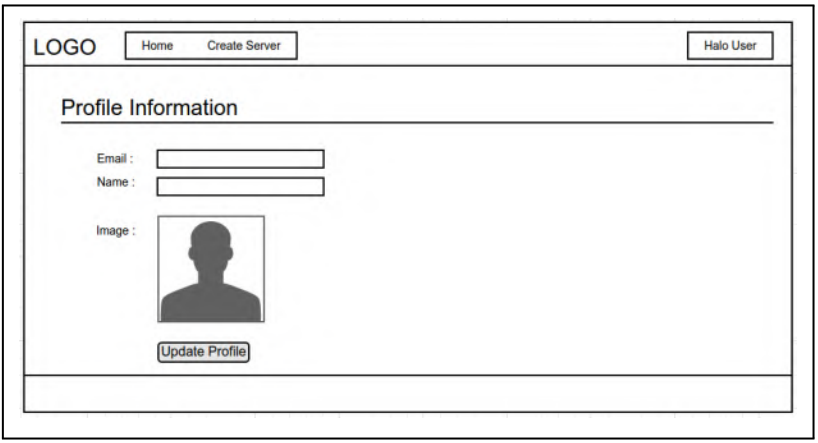


Gambar 3.10: Rancangan Antarmuka Dasbor Pengguna

Pengguna dapat membuat *server* peer baru dengan menekan tombol ”Buat Server” dan akan diarahkan ke halaman dasbor manajemen konten. Nama aplikasi *server* mengikuti nama identitas pengguna yang sedang masuk ke sistem saat itu.

3.5.1.3 Halaman Profil Pengguna

Seperti yang terlihat pada Gambar 3.11, halaman ini adalah halaman yang digunakan untuk mengubah detail akun dan menghapus akun pengguna. Informasi profil terdiri dari *email*, nama, gambar, dan *password*.



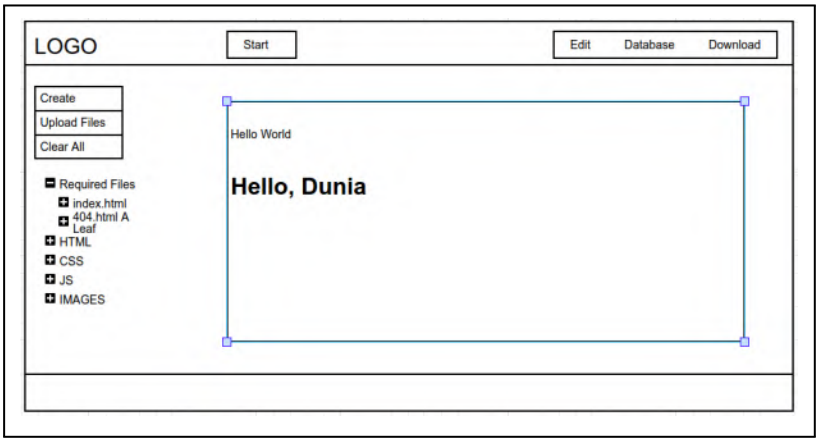
Gambar 3.11: Rancangan Antarmuka Profil Pengguna

3.5.2 Rancangan Antarmuka Manajemen Konten

Merupakan antarmuka yang digunakan untuk aktivitas *live coding* seperti pengimplementasian kode, pembuatan basis data, pembuatan rute aplikasi, dan mengunggah file gambar. Antarmuka manajemen konten terdiri dari, halaman dasbor konten, dasbor manajemen basis data, dasbor manajemen rute, dan dasbor manajemen gambar.

3.5.2.1 Halaman Dasbor Manajemen Konten

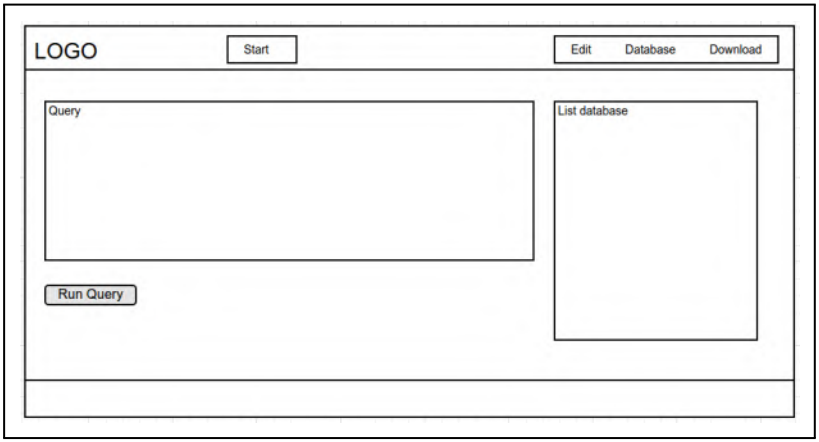
Pada Gambar 3.12 akan disediakan fungsionalitas untuk melakukan manajemen konten untuk melakukan pemrograman web berupa *editor* teks yang sudah terpasang di website. Fungsional pembuatan file baru, fasilitas pengunggahan file gambar.



Gambar 3.12: Rancangan Antarmuka Dasbor Konten

3.5.2.2 Halaman Dasbor Manajemen Basis Data

Seperti yang terlihat pada Gambar 3.13 akan disediakan fungsionalitas untuk memanajemen basis data aplikasi peer yang akan dibuat.

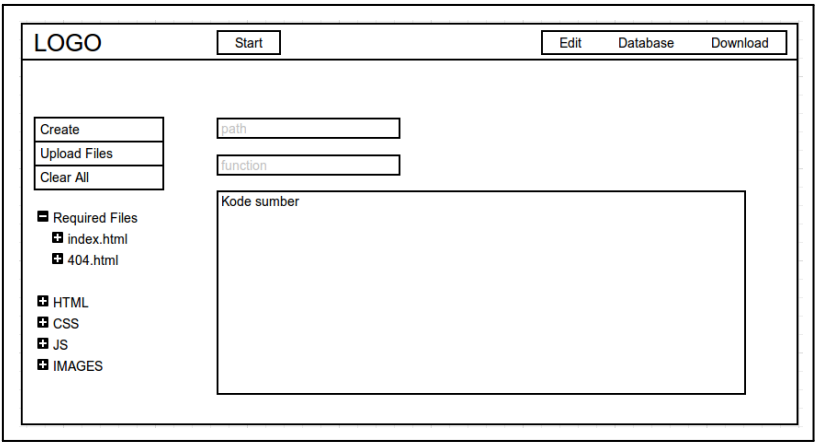


Gambar 3.13: Rancangan Antarmuka Dasbor Basis Data

Basis data yang digunakan adalah basis data berbasis file dengan format JSON yaitu TaffyDB.

3.5.2.3 Halaman Dasbor Manajemen Rute

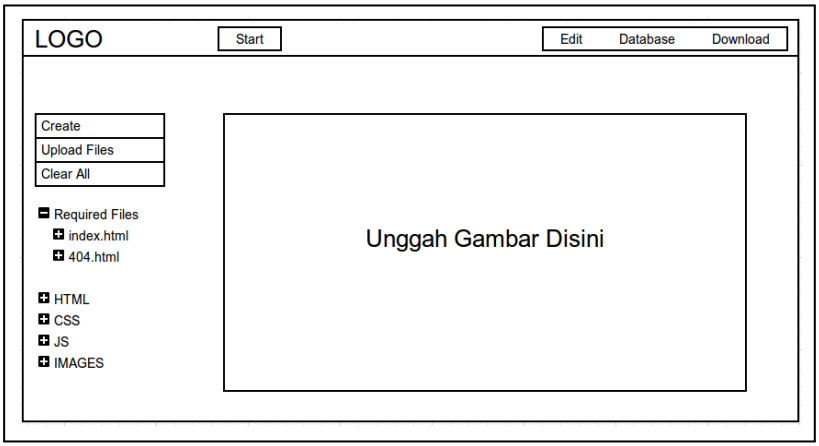
Pada Gambar 3.14 akan disediakan fungsionalitas untuk melakukan manajemen rute, manajemen rute digunakan untuk mengatur tautan `URL` aplikasi. Manajemen rute dilakukan dengan menentukan *path* rute dan fungsi. Selanjutnya pengguna dapat mengimplementasikan kode fungsi pada *editor* teks yang disediakan.



Gambar 3.14: Rancangan Antarmuka Dasbor Manajemen Rute

3.5.2.4 Halaman Dasbor Manajemen Gambar

Gambar 3.15 merupakan perancangan antarmuka untuk dasbor manajemen gambar, pada halaman ini pengguna dapat mengunggah gambar yang dibutuhkan untuk kemudian dipakai dalam aplikasi website yang dibuat. Jenis file yang didukung adalah .jpg, .png, .jpeg.



Gambar 3.15: Rancangan Antarmuka Dasbor Manajemen Gambar

BAB 4

IMPLEMENTASI

Bab ini membahas implementasi perancangan perangkat lunak dari aplikasi ini yang merupakan penerapan data, kebutuhan, dan alur sistem yang mengacu pada desain dan perancangan yang telah dibahas sebelumnya. Selain itu, bab ini juga membahas lingkungan pembangunan perangkat lunak yang digunakan dalam pembangunan sistem.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan komputer dengan spesifikasi Intel®Core™i3-3220 CPU @ 3.30GHz dengan memori 6 GB di Laboratorium Arsitektur dan Jaringan Komputer, Teknik Informatika ITS. Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut :

- Sistem Operasi Linux Ubuntu Server 16.04 LTS
- Desktop xfce4
- Editor teks vim
- Editor teks Sublime Text build 3114
- git versi 1.9.1 untuk pengolahan versi program
- NodeJS versi 5.0.0 untuk pengembangan aplikasi
- PeerJS versi 0.3.14 untuk pengembangan WebRTC
- MongoDB versi 3.2.6 untuk basis data
- Express versi 4.13.1 untuk web *server* halaman manajemen
- Paket LaTeX untuk pembuatan buku Tugas Akhir
- Peramban *web* Google Chrome versi 51.0.2704.84

4.2 Implementasi Perangkat Lunak

Implementasi pada sistem ini akan terbagi menjadi dua, yaitu bagian implementasi perangkat lunak dan implementasi antarmuka, implementasi perangkat lunak sendiri dibagi lagi

menjadi dua bagian, implementasi perangkat lunak manajemen konten dan implementasi perangkat lunak web manajemen pengguna. Setiap bagian dalam implementasi ini akan dijelaskan sesuai dengan urutan sistem berjalan.

4.2.1 Implementasi Manajemen Pengguna

Pada implementasi manajemen pengguna digunakan kerangka kerja Express berbasis Javascript yang dijalankan pada NodeJS, manajemen pengguna menggunakan basis data MongoDB sebagai media penyimpanan data pengguna.

4.2.1.1 Implementasi Rute Aplikasi

Implementasi rute aplikasi merupakan penjabaran dari desain rancangan rute pada Tabel 3.2, implementasi dan penjabarannya dijelaskan pada Tabel 4.1. Kode sumber implementasi rute disertakan pada halaman lampiran.

Tabel 4.1: Implementasi Rute Manajemen Pengguna

No	Rute	Metode	Hak akses	Masukan	Luaran	Langkah proses
1	`/`	GET	Tidak ada	-	Halaman utama aplikasi	Menerima data dari rute aplikasi, menampilkan *view*
2	`/login`	GET	Tidak ada	-	Halaman masuk aplikasi	Menerima data dari rute aplikasi, menampilkan *view* halaman masuk
3	`/signup`	GET	Tidak ada	-	Halaman pendaftaran aplikasi	Menerima data dari rute aplikasi, menampilkan *view* halaman pendaftaran
4	`/account`	GET	Pengguna	-	Halaman profil pengguna	Menerima data dari rute aplikasi, menampilkan *view* halaman pengguna

Tabel 4.1: Implementasi Rute Manajemen Pengguna

No	Rute	Metode	Hak akses	Masukan	Luaran	Langkah proses
5	`/login`	POST	Tidak ada	*Emai* dan *password* pengguna	Halaman dasbor pengguna	1. Menerima data *email* dan *password* dari halaman masuk. 2. Jika pengguna terdaftar maka akan diarahkan ke halaman dasbor pengguna.

Tabel 4.1: Implementasi Rute Manajemen Pengguna

No	Rute	Metode	Hak akses	Masukan	Luaran	Langkah proses
6	`/signup`	POST	Tidak ada	*Emai* dan *password* pengguna	Halaman masuk pengguna	1. Menerima data *email* dan *password* dari halaman pendaftaran. 2. Jika pengguna terdaftar maka akan diberikan pesan galat. 3. Jika pengguna belum terdaftar, maka data akan disimpan di basis data, dan diarhkan ke halaman masuk.

Tabel 4.1: Implementasi Rute Manajemen Pengguna

No	Rute	Metode	Hak akses	Masukan	Luaran	Langkah proses
7	`/account profile`	POST	Pengguna	Nama, *email* pengguna	Notifikasi	1. Menerima data Nama dan *password* dari halaman profil. 2. Melakukan perubahan basis data sesuai dengan baris dan kolom masukan. 3. Notifikasi data berhasil disimpan.

Tabel 4.1: Implementasi Rute Manajemen Pengguna

No	Rute	Metode	Hak akses	Masukan	Luaran	Langkah proses
8	`/account password`	POST	Pengguna	*Password* pengguna	Notifikasi	1. Menerima data *password* dari halaman profil. 2. Melakukan update *password* pada database sesuai dengan data pengguna yang masuk ke sistem. *3.* Notifikasi *password* berhasil diubah.

Tabel 4.1: Implementasi Rute Manajemen Pengguna

No	Rute	Metode	Hak akses	Masukan	Luaran	Langkah proses
9	`/account delete`	POST	Pengguna	ID Pengguna	Notifikasi, halaman utama	1. Menerima data ID pengguna. 2. Menghapus data pengguna di basis data sesuai dengan ID pengguna. 3. Data pengguna terhapus, sistem mengirimkan notifikasi, pengguna diarahkan ke halaman utama tanpa hak akses.

4.2.2 Implementasi Manajemen Konten

Pada implementasi manajemen konten digunakan PeerJS untuk memanajemen koneksi antar peramban yang selanjutnya disebut sebagai *Browser as Server* dan *Browser as Client* sehingga baris kode yang di buat oleh *Browser as Server* dapat diakses dan dijalankan di *Browser as Client* melalui arsitektur *peer-to-peer* menggunakan WebRTC.

4.2.2.1 Implementasi Dasbor

Proses pertama dalam sistem ini adalah penentuan ID Pengguna yang didapatkan dari basis data pengguna saat masuk ke sistem. Setiap pengguna akan mendapatkan ID acak yang unik dan ID ini akan digunakan sebagai ID *server*, kode sumber lengkap implementasi dasbor dapat dilihat di halaman Lampiran Kode Sumber B.1. Secara garis besar proses ini berjalan terlihat seperti pada Gambar 4.1

Prosedur 1 : Inisialisasi Dasbor

Input : id pengguna

```
1.  Inisialisasi
2.  IF id pengguna ada THEN
3.      RESPON proses file id pengguna
4.  ELSE
5.      RESPON proses file index.html
6.  END
```

Output : -

Gambar 4.1: *Pseudocode* Pembuatan Dasbor Manajemen Konten

Penjelasan dari *pseudocode* pembuatan dasbor manajemen konten adalah sebagai berikut:

1. Pengguna mengakses rute `/server/:id`.

2. Mengirimkan id untuk diproses
3. Jika id pengguna ada maka respon proses file sesuai id pengguna.
4. Jika tidak ada maka proses file index.html dengan id acak.

4.2.2.2 Implementasi Koneksi Peer

Gambar 4.2 menunjukkan *pseudocode* implementasi dari koneksi antar peer. Pada saat inisiasi pertama kali membuka halaman, pengguna akan mendapatkan identitas peer random ketika identitas peer tidak diatur.

Prosedur 2 : Inisialisasi Koneksi Peer

Input : -

```
1.  Load PeerJS client
2.  SEND port
3.  FUNCTION PeerServer(port)
4.     IF PeerServer tidak ada
5.        Buat PeerServer baru
6.        Inisialisasi WebSocket
7.        Set ID Peer
8.     ELSE
          Jalankan PeerServer
    ENDFUNCTION
```

Output : -

Gambar 4.2: *Pseudocode* Inisialisasi Koneksi Peer

Penjelasan dari *pseudocode* inisialisasi koneksi peer adalah sebagai berikut:

1. Melakukan load PeerJS yang ada pada aplikasi.
2. Mengirimkan port konfigurasi.
3. Menjalankan fungsi PeerServer dengan opsi port konfigurasi.

4. Jika PeerServer tidak ada maka akan dibuat PeerServer baru dengan inisialisasi WebSocket, kemudian mengatur ID Peer.
5. Jika PeerServer sudah ada maka jalankan PeerServer

Kode sumber lengkap untuk *pseudocode* inisialisasi koneksi peer dapat dilihat di Lampiran Kode Sumber B.2

4.2.2.3 Implementasi Pemroses AJAX Client

Gambar 4.3 menunjukkan *pseudocode* implementasi pemroses AJAX pada peramban.Penggunaan AJAX dilakukan untuk pemrosesan permintaan tanpa merubah `URI` peramban sehingga data tetap terproses tanpa melakukan penyegaran halaman.

Prosedur 3 : Pemrosesan AJAX Client

Input : SendEvent, SocketID

```
1.  Inisialisasi
2.  SEND SendEvent
3.  SEND SocketID
4.  FUNCTION AjaxClient(SendEvent, SocketID)
5.      SendEvent → SendEvent
6.      SocketIdFcn → socketIdFcn;
7.      Terima data Ajax
8.      Terima permintaan Ajax
9.      Proses permintaan
10. ENDFUNCTION
```

Output : -

Gambar 4.3: *Pseudocode* Pemroses AJAX CLient

Penjelasan dari *pseudocode* implementasi pemrosesan AJAX adalah sebagai berikut:

1. Inisialisasi permintaan pada aplikasi berupa SendEvent dan SocketID.

2. Menjalankan fungsi AjaxCLient dengan parameter inisialisasi.
3. Proses data Ajax yang diterima dan proses permintaan Ajax dari peer lain.

Kode sumber lengkap untuk *pseudocode* implenetasi pemroses AJAX dapat dilihat di Lampiran Kode Sumber B.3

4.2.2.4 Implementasi Pemroses Konten HTML

Gambar 4.4 menunjukkan *pseudocode* implementasi pemroses konten HTML pada peramban. Data mentah yang masuk harus diproses kembali menjadi format HTML yang baik agar dapat ditampilkan pada peramban.

```
Prosedur 3 : Pemrosesan Konten HTML
Input : data HTML
1.  Inisialisasi Kontainer
2.  CEK CompletionCallback
3.  Lakukan ScriptMapping
4.  FUNCTION ProsesHTML
5.      Proses Title
6.      Proses Gambar
7.      Proses Script
8.      Proses Stylesheet
9.      Proses Link
10.     Cek ProsesCompletion
    ENDFUNCTION
Output : Tampilan website
```

Gambar 4.4: *Pseudocode* Pemroses Konten HTML

Penjelasan dari *pseudocode* implementasi pemrosesan konten HTML adalah sebagai berikut:

1. Inisialisasi kontainer untuk menjalankan konten HTML.

2. Pengecekan CompletionCallback untuk melihat apakah ada data yang belum terproses.
3. Lakukan ScriptMapping untuk memetakan bagian - bagian data mentah yang dikirim.
4. Proses HTML dengan melakukan pemrosesan bagian - bagiannya antara lain Title, Gambar, Script, Stylesheet, dan Link.
5. Cek ProsesCompletion untuk memastikan proses selesai dilakukan.

Kode sumber lengkap untuk *pseudocode* implementasi pemroses konten HTML dapat dilihat di Lampiran Kode Sumber B.4

4.3 Implementasi Antarmuka Sistem

Berikut akan dijelaskan mengenai implementasi tampilan antarmuka dari sistem manajemen pengguna yang dibangun beserta tampilan dari manajemen konten. Antarmuka sistem disini dibuat untuk memudahkan interaksi pengguna dengan sistem melalui perintah-perintah yang disediakan sistem melalui tautan ataupun tampilan interaktif.

4.3.1 Implementasi Antarmuka Manajemen Pengguna

Implementasi antarmuka dari manajemen pengguna dibagi kedalam beberapa bagian yaitu, tampilan antarmuka halaman utama, tampilan halaman dasbor pengguna, dan tampilan halaman profil pengguna. Setiap bagian tampilan merepresentasikan fungsi masing - masing bagian dan perannya didalam sistem.

4.3.1.1 Implementasi Antarmuka Halaman Utama

Gambar 4.5: Implementasi Antarmuka Halaman Utama

Gambar 4.5 adalah implementasi halaman utama dari sistem manajemen web yang dibangun. Pada baris navigasi disediakan empat tautan. Tautan pertama untuk kembali ke halaman utama, tautan kedua untuk membuat *server*, tautan ketiga untuk masuk ke sistem, dan tautan keempat untuk mendaftarkan akun ke sistem. Tidak ada hak akses khusus untuk halaman ini.

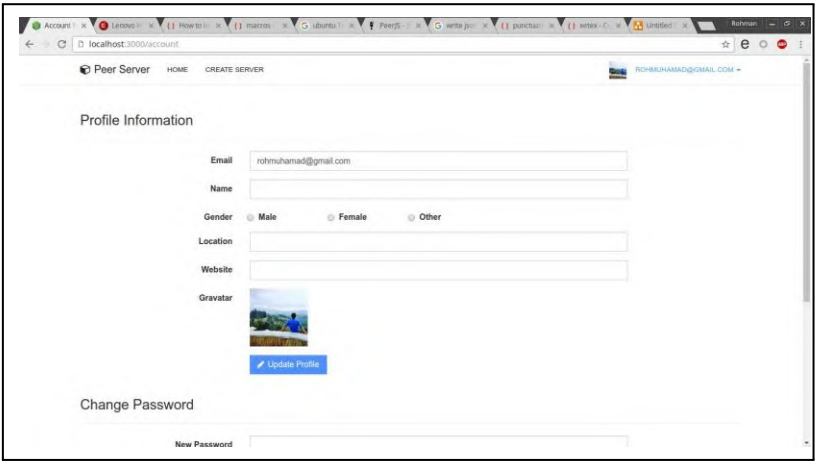
4.3.1.2 Implementasi Antarmuka Halaman Dasbor Pengguna

Gambar 4.6 adalah implementasi halaman dasbor pengguna digunakan untuk membuat *node server* baru dengan menekan menu Create Server. Setelah menu Create Server di klik, maka pengguna akan diarahkan ke halaman baru untuk manajemen konten dan berperan sebagai *server* selanjutnya *server* yang dibuat disebut *Browser as Server*. Hak akses halaman ini adalah level pengguna.

Gambar 4.6: Implementasi Antarmuka Halaman Dasbor Pengguna

4.3.1.3 Implementasi Antarmuka Halaman Profil Pengguna

Gambar 4.7 adalah implementasi halaman profil pengguna, digunakan untuk melakukan manajemen profil seperti merubah kata sandi, mengganti nama, dan menghapus akun.



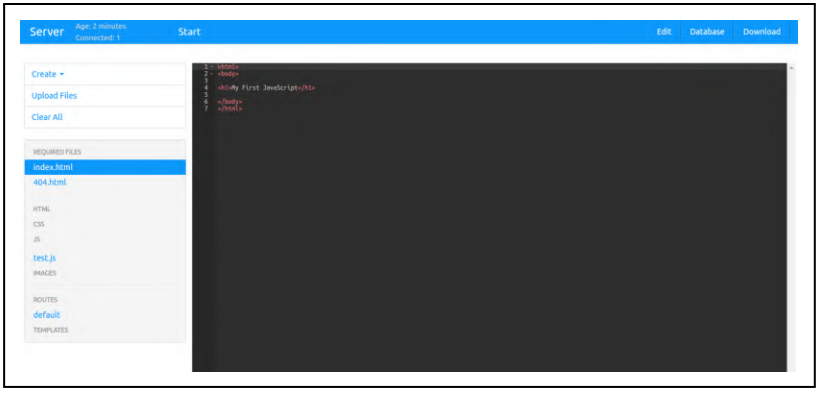
Gambar 4.7: Implementasi Antarmuka Halaman Profil Pengguna

4.3.2 Implementasi Antarmuka Manajemen Konten

Implementasi antarmuka dari manajemen konten dibagi kedalam beberapa bagian yaitu, tampilan antarmuka halaman dasbor konten, tampilan antarmuka halaman unggah, tampilan antarmuka halaman rute dan tampilan antarmuka halaman basis data.

4.3.2.1 Implementasi Antarmuka Halaman Dasbor Konten

Gambar 4.8 merupakan implementasi halaman dasbor konten, digunakan untuk melakukan pengaplikasian baris kode, pada halaman ini disediakan *editor* teks yang tertanam langsung di dalam sistem, pengguna dapat melakukan aktivitas *live coding* pada alat yang disediakan.



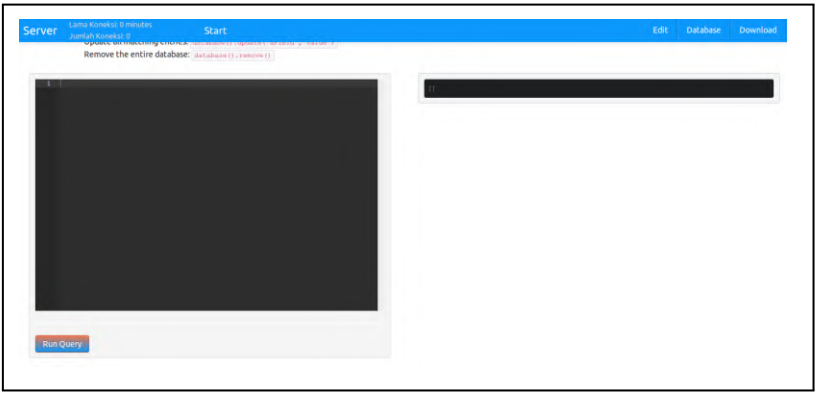
Gambar 4.8: Implementasi Antarmuka Halaman Dasbor Konten

4.3.2.2 Implementasi Antarmuka Halaman Manajemen Basis Data

Gambar 4.9 merupakan implementasi halaman manajemen basis data, basis data yang digunakan merupakan basis data NoSQL menggunakan `TaffyDB`. Pengguna dapat

memanfaatkan halaman manajemen basis data ini untuk melakukan operasi - operasi basis data sesuai dengan dokumentasi yang diberikan pada halaman website `TaffyDB`. Beberapa perintah operasi basis data yang dapat dilakukan antara lain:

1. Menambahkan item : `database.insert("type": "sometype", "aField": "someValue");`
2. Mencari satu item database: `return database("type": "sometype", "anotherField": "someValue").first()`
3. Update item sesuai kriteria: `database().update("aField", "value")`
4. Hapus basis data: `database().remove()`

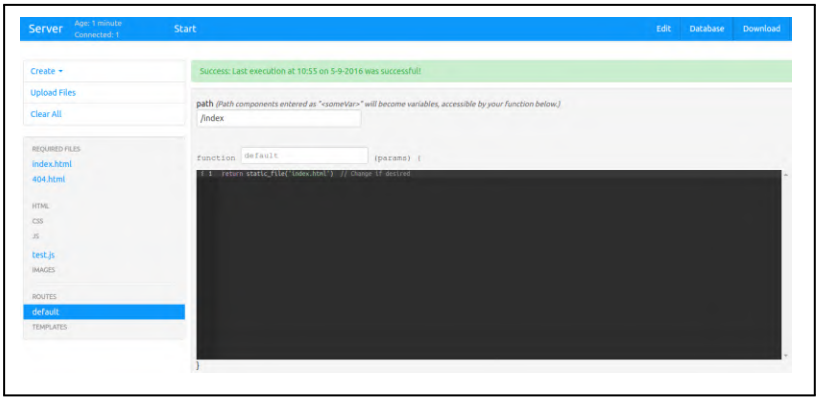


Gambar 4.9: Implementasi Antarmuka Halaman Manajemen Basis Data

4.3.2.3 Implementasi Antarmuka Halaman Manajemen Rute

Gambar 4.10 merupakan implementasi halaman manajemen rute aplikasi, digunakan untuk mengatur rute aplikasi layaknya *controller* pada sebuah sistem website berbasis MVC (*Model*

View Controller), pengguna dapat menentukan fungsi yang dijalankan untuk setiap rute yang dibuat.



Gambar 4.10: Implementasi Antarmuka Halaman Manajemen Rute

4.3.2.4 Implementasi Antarmuka Halaman Manajemen Gambar

Gambar 4.11 menunjukan antarmuka manajemen gambar, pada halaman ini pengguna dapat mengunggah gambar dengan melakukan *drag and drop* file gambar yang dibutuhkan, gambar akan diubah kedalam format `base64` untuk kemudian dikirim sebagai *string* dan diolah kembali di peramban tujuan.

Gambar 4.11: Implementasi Antarmuka Halaman Manajemen Gambar

LAMPIRAN A

INSTALASI PERANGKAT LUNAK

A.1 Instalasi Node JS

Ada banyak cara untuk menginstall NodeJS ke dalam komputer, salah satunya melalui kode sumber yang dapat diunduh pada halaman `https://nodejs.org/en/download/`. Setelah diunduh berkas perlu diekstrak untuk melanjutkan instalasi. Di dalam folder hasil ekstraksi, terdapat berkas `README.md` yang berisi langkah untuk menginstall NodeJS ke komputer. Semua langkah dilakukan melalui *command line* dan langkah installasinya adalah sebagai berikut

- Pindah ke folder dimana Node JS di ekstrak dengan perintah `cd {$pathdownload}/node-v4.2.3/`.
- Di dalam folder jalankan perintah berikut secara berurutan `./configure`, `make`, `make install`.
- Jika terdapat pesan *error* mengenai beberapa pustaka yang belum terinstal sebelumnya, jalankan perintah `sudo apt-get install -f`.
- Semua perintah dilakukan pada hak akses *root*.

Untuk memastikan bahwa Node JS sudah berjalan dan siap digunakan, dijalankan perintah dan dihasilkan output seperti pada Gambar A.1.

Selain menggunakan kompilasi sumber kode dari repositori NodeJS dapat juga digunakan installasi melalui terminal dengan menggunakan perintah `sudo apt-get install nodejs` pada sistem operasi Ubuntu. Untuk melengkapi fungsional NodeJS biasanya disertakan installasi manajer paket `npm` dengan mengetikkan pada terminal `sudo apt-get install npm`

Gambar A.1: Contoh Node JS Siap Digunakan

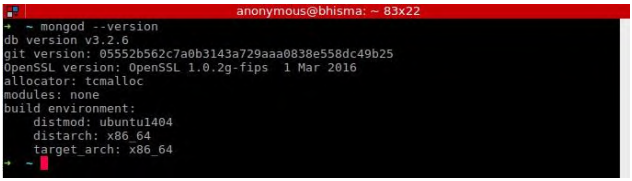
A.2 Installasi MongoDB

MongoDB menyediakan repositori tersendiri untuk mengunduh MongoDB. Setelah terkoneksi dengan repositori yang dimiliki MongoDB, instalasi dapat berlanjut dengan menggunakan perintah `apt-get`. Berikut ini langkah-langkah instalasi MongoDB.

- Tambahkan *public key* yang digunakan untuk manajemen paket pada sistem operasi dengan perintah `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10`.
- Buat file repositori untuk MongoDB pada sistem komputer dengan perintah `echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-`

`3.0.list`. Perintah ini disesuaikan dengan distro Linux yang digunakan.

- Perbaharui basis data manajemen paket dengan perintah `sudo apt-get update`.
- Install MongoDB ke komputer dengan perintah `sudo apt-get install -y mongodb-org`.
- Setelah instalasi selesai jalankan MongoDB dengan perintah `sudo service mongod start`.
- MongoDB siap digunakan dengan tampilan pada Gambar A.2.



Gambar A.2: Contoh MongoDB dan Daftar Database

LAMPIRAN B

KODE SUMBER

B.1 Server Manajemen Konten

Merupakan kode sumber untuk menjalankan *server* manajemen konten pada NodeJS.

Kode Sumber B.1: Kode Sumber Lengkap Server Manajemen Konten

```
1   var config = require('getconfig'),
2   http = require('http'),
3   fs = require('fs');
4   var app = require('express')();
5
6   /* Menjalankan peer server untuk aplikasi */
7   var PeerServer = require('peer').PeerServer;
8   var peerServer = new PeerServer({ port: 9000,
        debug: true });
9
10  /* Membuat file server untuk aplikasi */
11  var server = require('http').createServer(app);
12
13  var port = process.env.PORT || config.server.
        port || 5000;
14  /* Jalankan server pada port yang ditentukan*/
15  server.listen(port, function() {
16  console.log('Server running at port ' + port);
17  });
18
19  /* Static file mappings */
20  app.param('filename');
21
22  app.get('/server', function(req, res) {
23  res.sendfile(  dirname + '/server/index.html');
24  });
25
26  app.get('/server/:filename(*)', function(req,
        res) {
27  var filename =   dirname + '/server/' + req.
        params.filename;
28  if (fs.existsSync(filename)) {
29  res.sendfile(filename);
```

```
30      } else {
31      res.sendfile(  dirname + '/server/index.html');
32      }
33      });
34
35      app.get('/template/:filename(*)', function(req,
            res) {
36      res.sendfile( dirname + '/sample_servers/' +
            req.params.filename + '.zip');
37      });
38
39      app.get('/connect/:serverid(*)', function(req,
            res) {
40      var serverid = req.params.serverid;
41      res.sendfile(  dirname + '/client/index.html');
42      });
43
44      app.get('/client/:filename(*)', function(req,
            res) {
45      var filename = req.params.filename;
46      res.sendfile(  dirname + '/client/' + filename);
47      });
48
49      app.get('/shared/:filename(*)', function(req,
            res) {
50      var filename = req.params.filename;
51      res.sendfile(  dirname + '/shared/' + filename);
52      });
53
54      app.get('/home/:filename(*)', function(req, res)
            {
55      var filename = req.params.filename;
56      res.sendfile(  dirname + '/home/' + filename);
57      });
58
59
60      /* Cuma buat testing*/
61      app.get('/test_files/:filename(*)', function(req
            , res) {
62      var filename = req.params.filename;
```

```
63      res.sendfile( dirname + '/test_files/' +
            filename);
64      });
65
66      app.get("/", function(req, res) {
67      res.sendfile(  dirname + '/home/index.html');
68      })
69
70      if (config.uid) {
71      process.setuid(config.uid);
72      }
```

B.2 Inisialisasi Koneksi Peer

Merupakan kode sumber untuk inisialisasi koneksi peer dan mengatur koneksi antar dua peer.

Kode Sumber B.2: Kode Sumber Inisialisasi Koneksi Peer

```
1       var util = require('./util');
2       var restify = require('restify');
3       var http = require('http');
4       var EventEmitter = require('events').
            EventEmitter;
5       var WebSocketServer = require('ws').Server;
6       var url = require('url');
7
8       function PeerServer(options) {
9       if (!(this instanceof PeerServer)) return new
            PeerServer(options);
10      EventEmitter.call(this);
11
12      this._app = restify.createServer();
13      this._httpServer = this._app;
14
15      this._options = util.extend({
16        port: 80,
17        debug: false,
18        timeout: 5000,
19        key: 'peerjs',
```

```
20          ip_limit: 5000,
21          concurrent_limit: 5000
22          }, options);
23
24        util.debug = this._options.debug;
25
26        this._clients = {}; //client terkoneksi
27
28        //Pesan tunggu untuk peer lain.
29        this._outstanding = {};
30
31        // Inisialisasi WebSocket.
32        this._initializeWSS();
33
34        // Inisialisasi rute HTTP
35        this._initializeHTTP();
36
37        // Tandai ip pengguna yang masuk bersama sama
38        this._ips = {};
39
40        this._setCleanupIntervals();
41
42        };
43
44        util.inherits(PeerServer, EventEmitter);
45
46
47        /** fungsi inisialisasi websocket. */
48        PeerServer.prototype._initializeWSS = function()
              {
49          var self = this;
50        // Membuat server WebSocket.
51        this._wss = new WebSocketServer({ path: '/peerjs
              ', server: this._httpServer });
52
53        this._wss.on('connection', function(socket) {
54        var query = url.parse(socket.upgradeReq.url,
              true).query;
55          var id = query.id;
56          var token = query.token;
57          var key = query.key;
```

```
58          var ip = socket.upgradeReq.socket.
                remoteAddress;
59
60          if (!id || !token || !key) {
61          socket.send(JSON.stringify({ type: 'ERROR',
                payload: { msg: 'No id, token, or key
                supplied to websocket server' } }));
62          socket.close();
63          return;
64        }
65
66        if (!self._clients[key] || !self._clients[key][
              id]) {
67          self._checkKey(key, ip, function(err) {
68          if (!err) {
69          if (!self._clients[key][id]) {
70          self._clients[key][id] = { token: token, ip:
                ip };
71          self. ips[ip]++;
72          socket.send(JSON.stringify({ type: 'OPEN' }));
73          }
74          self._configureWS(socket, key, id, token);
75          } else {
76          socket.send(JSON.stringify({ type: 'ERROR',
                payload: { msg: err } }));
77          }
78          });
79          } else {
80          self._configureWS(socket, key, id, token);
81          }
82          });
83        };
84
85        PeerServer.prototype._configureWS = function(
              socket, key, id, token) {
86          var self = this;
87          var client = this._clients[key][id];
88
89          if (token === client.token) {
90          client.socket = socket;
91          if (client.res) {
```

```
92          client.res.end();
93          }
94          } else {
95          socket.send(JSON.stringify({ type: 'ID-TAKEN',
                payload: { msg: 'ID is taken' } }));
96          socket.close();
97          return;
98          }
99
100       this._processOutstanding(key, id);
101
102         socket.on('close', function() {
103         util.log('Socket closed:', id);
104         if (client.socket == socket) {
105         self._removePeer(key, id);
106         }
107       });
108
109       socket.on('message', function(data) {
110         try {
111         var message = JSON.parse(data);
112
113         switch (message.type) {
114         case 'LEAVE':
115         if (!message.dst) {
116         self._removePeer(key, id);
117         break;
118         }
119         case 'CANDIDATE':
120         case 'OFFER':
121         case 'ANSWER':
122
123         self._handleTransmission(key, {
124         type: message.type,
125         src: id,
126         dst: message.dst,
127         payload: message.payload
128         });
129         break;
130         default:
131         util.prettyError('Message unrecognized');
```

```
132         }
133         } catch(e) {
134         throw e;
135         util.log('Invalid message', data);
136         }
137         });
138       }
139
140
141       PeerServer.prototype._checkKey = function(key,
              ip, cb) {
142         if (key == this._options.key) {
143         if (!this._clients[key]) {
144         this._clients[key] = {};
145         }
146         if (!this._outstanding[key]) {
147         this._outstanding[key] = {};
148         }
149         if (!this._ips[ip]) {
150         this._ips[ip] = 0;
151         }
152
153       if (Object.keys(this._clients[key]).length >=
              this._options.concurrent_limit) {
154         cb('Server has reached its concurrent user
              limit');
155         return;
156         }
157         if (this._ips[ip] >= this._options.ip_limit) {
158         cb(ip + ' has reached its concurrent user
              limit');
159         return;
160         }
161         cb(null);
162         } else {
163         cb('Invalid key provided');
164         }
165       }
166
167       PeerServer.prototype._initializeHTTP = function
              () {
```

```
168         var self = this;
169
170         this._app.use(restify.bodyParser({ mapParams:
                false }));
171         this._app.use(restify.queryParser())
172         this._app.use(util.allowCrossDomain);
173
174         this._app.get('/:key/id', function(req, res,
                next) {
175         res.contentType = 'text/html';
176         res.send(self._generateClientId(req.params.key
                ));
177         return next();
178         });
179
180         post an ID.
181         this._app.post('/:key/:id/:token/id', function
                (req, res, next) {
182         var id = req.params.id;
183         var token = req.params.token;
184         var key = req.params.key;
185         var ip = req.ip;
186
187         if (!self._clients[key] || !self._clients[key
                ][id]) {
188         self._checkKey(key, ip, function(err) {
189         if (!err && !self._clients[key][id]) {
190         self._clients[key][id] = { token: token, ip:
                ip };
191         self._ips[ip]++;
192         self._startStreaming(res, key, id, token, true
                );
193         } else {
194         res.send(JSON.stringify({ type: 'HTTP-ERROR'
                }));
195         }
196         });
197         } else {
198         self._startStreaming(res, key, id, token);
199         }
200         return next();
```

```
201       });
202
203
204       var handle = function(req, res, next) {
205       var key = req.params.key;
206       var id = req.params.id;
207
208       var client;
209         if (!self._clients[key] || !(client = self.
              _clients[key][id])) {
210         if (req.params.retry) {
211         res.send(401);
212         } else {
213         req.params.retry = true;
214         setTimeout(handle, 25, req, res);
215         }
216         return;
217         }
218
219         if (req.params.token !== client.token) {
220         res.send(401);
221         return;
222         } else {
223         self._handleTransmission(key, {
224         type: req.body.type,
225         src: id,
226         dst: req.body.dst,
227         payload: req.body.payload
228         });
229         res.send(200);
230         }
231         return next();
232         };
233
234         this._app.post('/:key/:id/:token/offer',
              handle);
235
236         this._app.post('/:key/:id/:token/candidate',
              handle);
237
```

```
238         this._app.post('/:key/:id/:token/answer',
                handle);
239
240         this._app.post('/:key/:id/:token/leave',
                handle);
241
242         this._httpServer.listen(this._options.port);
243
244         };
245
246
247         PeerServer.prototype._startStreaming =
                function(res, key, id, token, open) {
248         var self = this;
249
250         res.writeHead(200, {'Content-Type': '
                application/octet-stream'});
251
252         var pad = '00';
253         for (var i = 0; i < 10; i++) {
254         pad += pad;
255         }
256         res.write(pad + '\n');
257
258         if (open) {
259         res.write(JSON.stringify({ type: 'OPEN' }) + '
                \n');
260         }
261
262         var client = this._clients[key][id];
263
264         if (token === client.token) {
265         // Client already exists
266         res.on('close', function() {
267         if (client.res === res) {
268         if (!client.socket) {
269
270         self._removePeer(key, id);
271         return;
272         }
273         delete client.res;
```

```
274         }
275         });
276         client.res = res;
277         this._processOutstanding(key, id);
278         } else {
279
280         res.end(JSON.stringify({ type: 'HTTP-ERROR' })
                );
281         }
282         };
283
284         PeerServer.prototype._pruneOutstanding =
                function() {
285         var keys = Object.keys(this._outstanding);
286         for (var k = 0, kk = keys.length; k < kk; k +=
                 1) {
287         var key = keys[k];
288         var dsts = Object.keys(this._outstanding[key])
                ;
289         for (var i = 0, ii = dsts.length; i < ii; i +=
                 1) {
290         var offers = this._outstanding[key][dsts[i]];
291         var seen = {};
292         for (var j = 0, jj = offers.length; j < jj; j
                += 1) {
293         var message = offers[j];
294         if (!seen[message.src]) {
295         this._handleTransmission(key, { type: 'EXPIRE'
                , src: message.dst, dst: message.src });
296         seen[message.src] = true;
297         }
298         }
299         }
300         this._outstanding[key] = {};
301         }
302       };
303
304       PeerServer.prototype._setCleanupIntervals =
              function() {
305         var self = this;
306
```

```
307         setInterval(function() {
308         var keys = Object.keys(self. ips);
309         for (var i = 0, ii = keys.length; i < ii; i +=
                1) {
310         var key = keys[i];
311         if (self._ips[key] == 0) {
312         delete self._ips[key];
313         }
314         }
315         }, 600000);
316
317         setInterval(function() {
318         self._pruneOutstanding();
319         }, 5000);
320       };
321
322       PeerServer.prototype._processOutstanding =
              function(key, id) {
323         var offers = this._outstanding[key][id];
324         if (!offers) {
325         return;
326         }
327         for (var j = 0, jj = offers.length; j < jj; j
              += 1) {
328         this._handleTransmission(key, offers[j]);
329         }
330         delete this._outstanding[key][id];
331       };
332
333       PeerServer.prototype._removePeer = function(key,
            id) {
334         if (this._clients[key] && this._clients[key][
              id]) {
335         this._ips[this._clients[key][id].ip]--;
336         delete this._clients[key][id];
337         }
338         };
339
340       PeerServer.prototype._handleTransmission =
              function(key, message) {
341         var type = message.type;
```

```
342         var src = message.src;
343         var dst = message.dst;
344         var data = JSON.stringify(message);
345 
346         var destination = this._clients[key][dst];
347 
348         if (destination) {
349         try {
350         util.log(type, 'from', src, 'to', dst);
351         if (destination.socket) {
352         destination.socket.send(data);
353         } else if (destination.res) {
354         data += '\n';
355         destination.res.write(data);
356         } else {
357         throw "Peer dead"
358         }
359         } catch (e) {
360         util.prettyError(e);
361         this._removePeer(key, dst);
362         this._handleTransmission(key, {
363         type: 'LEAVE',
364         src: dst,
365         dst: src
366         });
367         }
368         } else {
369         if (type !== 'LEAVE' && type !== 'EXPIRE' &&
                !!dst) {
370         var self = this;
371         if (!this._outstanding[key][dst]) {
372         this._outstanding[key][dst] = [];
373         }
374         this._outstanding[key][dst].push(message);
375         } else if (type === 'LEAVE' && !dst) {
376         this._removePeer(key, src);
377         } else {
378         }
379         }
380       };
381 
```

```
382       PeerServer.prototype._generateClientId =
              function(key) {
383         var clientId = util.randomId();
384         if (!this._clients[key]) {
385         return clientId;
386         }
387         while (!!this._clients[key][clientId]) {
388         clientId = util.randomId();
389         }
390         return clientId;
391         };
392
393         exports.PeerServer = PeerServer;
```

B.3 Pemroses AJAX Client

Merupakan kode sumber pemroses AJAX *Client*, digunakan untuk memproses permintaan transaksi data melalui AJAX pada peramban.

Kode Sumber B.3: Kode Sumber Pemroses AJAX Client

```
1       (function() {
2       var   bind = function(fn, me){ return function()
            { return fn.apply(me, arguments); }; };
3
4       window.AjaxClient = (function() {
5       function AjaxClient(sendEvent, socketIdFcn) {
6       this.sendEvent = sendEvent;
7       this.socketIdFcn = socketIdFcn;
8       this.receiveAjax =   bind(this.receiveAjax, this
            );
9       this.requestAjax =   bind(this.requestAjax, this
            );
10      this.outstandingRequests = {};
11      }
12
13      AjaxClient.prototype.requestAjax = function(path
            , options, successCallback, errorCallback) {
14      var data, requestId;
```

```
15      requestId = Math.random().toString(36).substr(2,
            14);
16      if (typeof callback !== "undefined" && typeof
            callback !== "function") {
17      console.error("error: callback is not a function
            !");
18      return;
19      }
20      this.outstandingRequests[requestId] = {
21      "successCallback": successCallback,
22      "errorCallback": errorCallback,
23      "timestamp": new Date().getTime()
24      };
25      data = {
26      "filename": path,
27      "socketId": this.socketIdFcn(),
28      "requestId": requestId,
29      "options": options,
30      "type": "ajax"
31      };
32      return this.sendEvent("requestFile", data);
33      };
34
35      AjaxClient.prototype.receiveAjax = function(data
            ) {
36      var request;
37      if (!data.requestId) {
38      console.error("received AJAX response with no
            request ID");
39      return;
40      }
41      request = this.outstandingRequests[data.
            requestId];
42      if (!request || typeof request === "undefined")
            {
43      console.error("received ajax response for a
            nonexistent request id");
44      return;
45      }
46      delete this.outstandingRequests[data.requestId];
47      if (!data.errorThrown) {
```

```
48      return request.successCallback(data.fileContents
            );
49      } else {
50      return request.errorCallback({}, data.textStatus
            , data.errorThrown);
51      }
52      };
53
54      return AjaxClient;
55
56      })();
57
58      }).call(this);
```

B.4 Pemroses Konten HTML

Merupakan kode sumber pemroses konten HTML, digunakan untuk memproses konten HTML seperti kode HTML, CSS, JS, dan gambar pada peramban.

Kode Sumber B.4: Kode Sumber Pemroses Konten HTML

```
1   (function() {
2   var bind = function(fn, me){ return function(){
        return fn.apply(me, arguments); }; };
3
4   window.HTMLProcessor = (function() {
5   function HTMLProcessor(sendEvent,
        setDocumentElementInnerHTML, getIDFn) {
6   this.sendEvent = sendEvent;
7   this.setDocumentElementInnerHTML =
        setDocumentElementInnerHTML;
8   this.getIDFn = getIDFn;
9   this.checkForProcessCompletion = bind(this.
        checkForProcessCompletion, this);
10  this.handleSupportingFile = bind(this.
        handleSupportingFile, this);
11  this.handleFileForElem = bind(this.
        handleFileForElem, this);
12  this.receiveFile =   bind(this.receiveFile, this);
```

```
13      this.removeTrailingSlash = bind(this.
            removeTrailingSlash, this);
14      this.requestFile =   bind(this.requestFile, this);
15      this.isInternalFile =   bind(this.isInternalFile,
            this);
16      this.processElementsWithAttribute = bind(this.
            processElementsWithAttribute, this);
17      this.triggerOnParentString =   bind(this.
            triggerOnParentString, this);
18      this.processLinks =   bind(this.processLinks, this
            );
19      this.processStyleSheets = bind(this.
            processStyleSheets, this);
20      this.processScripts =   bind(this.processScripts,
            this);
21      this.processImages =   bind(this.processImages,
            this);
22      this.processImageAsHTML = bind(this.
            processImageAsHTML, this);
23      this.processTitle =   bind(this.processTitle, this
            );
24      this.processHTML =   bind(this.processHTML, this);
25      this.requestedFilenamesToElement = {};
26      this.container = null;
27      this.completionCallback = null;
28      }
29
30      HTMLProcessor.prototype.processHTML = function(
            html, completionCallback) {
31      var container;
32      this.completionCallback = completionCallback;
33      this.scriptMapping = {};
34      container = document.createElement("html");
35      container.innerHTML = html.replace(/<\/?html>/g, "
            ");
36      this.container = $(container);
37      this.processTitle();
38      this.processImages();
39      this.processScripts();
40      this.processStyleSheets();
41      this.processLinks();
```

```
42      return this.checkForProcessCompletion();
43      };
44
45      HTMLProcessor.prototype.processTitle = function()
            {
46      var elements,
47      _this = this;
48      elements = this.container.find("title");
49      return elements.each(function(index, el) {
50      var $el;
51      $el = $(el);
52      return document.title = $el.text();
53      });
54      };
55
56      HTMLProcessor.prototype.processImageAsHTML =
            function(html, completionCallback) {
57      var container, img;
58      this.completionCallback = completionCallback;
59      container = document.createElement("html");
60      this.container = $(container);
61      img = "<img style='text-align:center; position:
            absolute; margin:auto; top:0;right:0;bottom:0;
            left:0;' ";
62      img += " src='" + html + "' />";
63      this.container.append($(img));
64      return this.completionCallback(this.container[0].
            outerHTML);
65      };
66
67      HTMLProcessor.prototype.processImages = function()
            {
68      return this.processElementsWithAttribute(this.
            container.find("img[src]"), "src", "image");
69      };
70
71      HTMLProcessor.prototype.processScripts = function
            () {
72      return this.processElementsWithAttribute(this.
            container.find("script[src]"), "src", "script"
            );
```

```
73      };
74
75      HTMLProcessor.prototype.processStyleSheets =
            function() {
76      var elements;
77      elements = this.container.find("link[rel=\"
            stylesheet\"]");
78      elements.add(this.container.find("link[rel=\'
            stylesheet\']"));
79      return this.processElementsWithAttribute(elements,
             "href", "stylesheet");
80      };
81
82      HTMLProcessor.prototype.processLinks = function()
            {
83      var elements,
84      _this = this;
85      elements = this.container.find("a[href]");
86      return elements.each(function(index, el) {
87      var $el, href;
88      $el = $(el);
89      href = $el.attr("href");
90      if (href[0] === "#") {
91
92      } else if (_this.isInternalFile(href)) {
93      return $el.attr("onclick", _this.
            triggerOnParentString("relativeLinkClicked",
            href));
94      } else {
95      return $el.attr("target", "_blank");
96      }
97      });
98      };
99
100     HTMLProcessor.prototype.triggerOnParentString =
            function(eventName, href) {
101     return "javascript:top.$(top.document).trigger('"
            + eventName + "', ['" + href + "']);return
            false;";
102     };
103
```

```
104     HTMLProcessor.prototype.
            processElementsWithAttribute = function(
            elements, attrSelector, type) {
105     var _this = this;
106     return elements.each(function(index, el) {
107     var $el, filename;
108     $el = $(el);
109     filename = $el.attr(attrSelector);
110     if (_this.isInternalFile(filename)) {
111     if (filename in _this.requestedFilenamesToElement)
            {
112     return _this.requestedFilenamesToElement[filename
            ].push($el);
113     } else {
114     _this.requestedFilenamesToElement[filename] = [$el
            ];
115     return _this.requestFile(filename, type);
116     }
117     }
118     });
119     };
120
121     HTMLProcessor.prototype.isInternalFile = function(
            filename) {
122     if (filename[0] !== "#" && filename.match(/(?:
            https?:\/\/)|(?:data:)/) === null) {
123     return true;
124     }
125     return false;
126     };
127
128     HTMLProcessor.prototype.requestFile = function(
            filename, type) {
129     var data;
130     data = {
131     "filename": filename,
132     "socketId": this.getIDFn(),
133     "type": type
134     };
135     return this.sendEvent("requestFile", data);
136     };
```

```
137
138     HTMLProcessor.prototype.removeTrailingSlash =
            function(str) {
139     if (!str || str === "") {
140     return str;
141     }
142     if (str.charAt(str.length - 1) === "/") {
143     return str.substr(0, str.length - 1);
144     }
145     return str;
146     };
147
148     HTMLProcessor.prototype.receiveFile = function(
            data) {
149     var fileContents, fileType, filename, type;
150     filename = this.removeTrailingSlash(data.filename)
            ;
151     fileContents = data.fileContents;
152     type = data.type;
153     fileType = data.fileType;
154     if (type === "alink" || type === "backbutton" ||
            type === "initialLoad" || type === "submit") {
155     return this.setDocumentElementInnerHTML({
156     "fileContents": data.fileContents,
157     "filename": filename,
158     "fileType": fileType
159     }, type);
160     } else {
161     return this.handleSupportingFile(data, filename,
            fileContents, type, fileType);
162     }
163     };
164
165     HTMLProcessor.prototype.handleFileForElem =
            function($element, data, filename,
            fileContents) {
166     if (!$element) {
167     console.error("element is null: " + filename);
168     return;
169     }
```

```
170     if ($element.attr("src") && $element[0].tagName
            === "IMG") {
171     return $element.attr("src", fileContents);
172     } else if ($element.attr("src") && $element[0].
            tagName === "SCRIPT") {
173     $element.removeAttr("src");
174     $element.attr("todo-replace", "replace");
175     this.scriptMapping[data.filename] = fileContents;
176     return $element.append(data.filename);
177     } else if ($element[0].tagName === "LINK") {
178     return $element.replaceWith("<style>" +
            fileContents + "</style>");
179     } else {
180     console.log("unknown element type, could not be
            processed:");
181     return console.log($element);
182     }
183     };
184
185     HTMLProcessor.prototype.handleSupportingFile =
            function(data, filename, fileContents, type,
            fileType) {
186     var $element, elems, _i, _len;
187     elems = this.requestedFilenamesToElement[filename
            ];
188     if (!elems) {
189     console.error("received file not in request list:
            " + filename);
190     return;
191     }
192     for (_i = 0, _len = elems.length; _i < _len; _i++)
             {
193     $element = elems[_i];
194     this.handleFileForElem($element, data, filename,
            fileContents);
195     }
196     delete this.requestedFilenamesToElement[filename];
197     return this.checkForProcessCompletion();
198     };
199
```

```
200     HTMLProcessor.prototype.checkForProcessCompletion
            = function() {
201     if (Object.keys(this.requestedFilenamesToElement).
            length === 0 && this.completionCallback) {
202     console.log("EXECUTING COMPLETION CALLBACK");
203     return this.completionCallback(this.container[0].
            outerHTML, this.scriptMapping);
204     }
205     };
206
207     return HTMLProcessor;
208
209     })();
210
211     }).call(this);
```

B.5 Inisialisasi Basis Data Lokal

Merupakan kode sumber pemroses basis data JSON, digunakan untuk memproses basis data pada penyimpanan lokal.

Kode Sumber B.5: Kode Sumber Inisialisasi Basis Data Lokal

```
1
2       (function() {
3       var _ref,
4       __bind = function(fn, me){ return function(){
            return fn.apply(me, arguments); }; },
5       __hasProp = {}.hasOwnProperty,
6       __extends = function(child, parent) { for (var key
             in parent) { if ( hasProp.call(parent, key))
             child[key] = parent[key]; } function ctor() {
             this.constructor = child; } ctor.prototype =
            parent.prototype; child.prototype = new ctor()
            ; child.  super___= parent.prototype; return
            child; };
7
8       window.UserDatabase = (function(_super) {
9       __extends(UserDatabase, _super);
10
```

```
11      function UserDatabase() {
12      this.onDBChange =   bind(this.onDBChange, this);
13      this.clear =   bind(this.clear, this);
14      this.runQuery =   bind(this.runQuery, this);
15      this.fromJSONArray =   bind(this.fromJSONArray,
            this);
16      this.toString =   bind(this.toString, this);
17      this.initLocalStorage = bind(this.
            initLocalStorage, this);
18      _ref = UserDatabase.  super  .constructor.apply(
            this, arguments);
19      return _ref;
20      }
21
22      UserDatabase.prototype.initialize = function() {
23      this.database = TAFFY();
24      return this.database.settings({
25      onDBChange: this.onDBChange
26      });
27      };
28
29      UserDatabase.prototype.initLocalStorage = function
            (namespace) {
30      this.database.store(namespace + "-UserDatabase");
31      return this.trigger("initLocalStorage");
32      };
33
34      UserDatabase.prototype.toString = function(pretty)
             {
35      if (pretty) {
36      return JSON.stringify(this.database().get(), null,
             4);
37      }
38      return this.database().stringify();
39      };
40
41      UserDatabase.prototype.fromJSONArray = function(
            array) {
42      return this.database.insert(array);
43      };
44
```

```
45      UserDatabase.prototype.runQuery = function(query)
            {
46      var code;
47      code = "(function(database) { " + query + " }).
            call(null, this.database)";
48      return eval(code);
49      };
50
51      UserDatabase.prototype.clear = function() {
52      return this.database().remove();
53      };
54
55      UserDatabase.prototype.onDBChange = function() {
56      return this.trigger("onDBChange");
57      };
58
59      return UserDatabase;
60
61      })(Backbone.Model);
62
63      }).call(this);
```

LAMPIRAN C

KUISIONER

C.1 Hasil Kuisioner

Berikut adalah hasil kuisioner terhadap 10 responden setelah menggunakan aplikasi Tugas Akhir ini. Kuisioner tersebut memiliki 8 pernyataan dengan bobot sebagai berikut:

- **Sangat setuju** = 5
- **Setuju** = 4
- **Biasa** = 3
- **Tidak setuju** = 2
- **Sangat Tidak Setuju** = 1

Berikut adalah rangkuman pernyataan kuisioner yang diberikan kepada responden:

1. Saya sudah terbiasa melakukan pemrograman web.
2. Saya selalu memakai IDE *(Integrated Development Environtment)* untuk membuat aplikasi website.
3. Saya baru pertama kali menggunakan IDE seperti pada aplikasi ini.
4. Aplikasi mudah digunakan.
5. Saya mengutamakan kecepatan dalam menggunakan aplikasi berbasis web.
6. Saya puas dengan kecepatan aplikasi ini.
7. Saya pernah bermasalah dengan kompatibilitas tools pemrograman web.
8. Saya bisa menjalankan aplikasi ini di perangkat saya dengan lancar.

Hasil lengkap kuisioner sesuai dengan pernyataan diatas dapat dilihat pada Tabel C.1.

Tabel C.1: Hasil Kuisioner

No	Atribut	Responden ke-										
		1	2	3	4	5	6	7	8	9	10	Rata - rata
1	Pernyataan 1	5	5	5	5	5	5	5	5	5	5	5
2	Pernyataan 2	5	4	4	5	5	5	4	5	4	4	4.5
3	Pernyataan 3	5	5	5	5	5	4	4	5	5	4	4.7
4	Pernyataan 4	3	3	4	4	3	3	4	5	3	4	3.6
5	Pernyataan 5	5	5	5	5	5	5	5	5	5	5	5
6	Pernyataan 6	2	3	3	2	3	2	2	3	2	3	2.5
7	Pernyataan 7	4	4	5	5	5	3	4	4	4	3	4.1
8	Pernyataan 8	4	5	5	4	5	4	5	5	5	5	4.7
-	**Rata - rata**	4.12	4.25	4.50	4.37	4.50	3.87	4.12	4.62	4.12	4.12	4.26

BAB 6

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Peramban web yang mendukung teknologi WebRTC memungkinkan untuk dikondisikan sebagai *server* layanan penyedia konten untuk aktivitas *live coding* dengan memanfaatkan fungsi Data Channel pada WebRTC
2. Skema layanan pengaturan distribusi konten antar peramban dapat dibuat dengan mengkombinasikan PeerJS yang menyediakan fitur STUN *server* sebagai jalur distribusi antar peer.
3. Dengan menggunakan bahasa Javascript, dapat dibangun antarmuka *editor* teks untuk keperluan pengetesan baris kode pemrograman web dan dapat diakses tanpa masalah kompatibilitas pada arsitektur perangkat komputer yang mendukung instalasi peramban Google Chrome versi 40 keatas.
4. Beban permintaan pada arsitektur *peer-to-peer* dibebankan langsung pada peer tujuan meskipun diakses melalui alamat *server* penyedia layanan. Dalam hal performa, arsitektur *peer-to-peer* sepuluh kali lebih lambat dalam menyajikan konten daripada arsitektur *client-server* pada umumnya.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Menambahkan fitur installasi dan pengembangan untuk kerangka kerja berbasis Javascript seperti Express, Ionic, Meteor, dsb.
- Pengembangan teknologi *peer-to-peer* dengan memanfaatkan DHT(*Distributed Hash table*) torrent untuk website, sehingga beban konten dapat dibagi rata, tidak hanya satu tujuan peer.

DAFTAR PUSTAKA

[1] Collins. N, McLean. A, Rohrhuber, and Ward. J, "Live Coding in Laptop Performance", Organized Sound 8(3), 2003, pp. 321-322.

[2] Loreto, S., & Romano, S. P, "Real-Time Communication with WebRTC", California: O'Reilly Media Inc,2014.

[3] 3scale Networks S.L, "What is an API? Your Guide to the Internet Business (R)evolution". 3scale, 4, 2011.

[4] Joyent, Inc, "About Node.js", [Online], https://nodejs.org/en/about/, diakses tanggal 01 April 2015.

[5] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs", IEEE Computer Society Issue, vol. 6, pp. 80-83, 2010.

[6] IETF, "Web RTC Data Channel", [Online] https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-13#section-6.4, diakses tanggal 20 Juni 2015.

[7] Peerjs, "Peejs-Server", [Online],http://peerjs.com diakses tanggal 27 Mei 2015.

[8] BackboneJS, "Backbone.js", [Online], https://backbonejs.org/ diakses tanggal 24 Juni 2015.

[9] HandlebarJS, "Handlebar.js", [Online], http://handlebarsjs.com/ diakses tanggal 24 Juni 2015.

[10] Ming Zhang, Jing Zhang, Wei Zheng, Feiran Hu, Ge Zhuang, "A Self-description Data Framework for Tokamak Control System Design", Fusion Engineering and Design, p. 5, 2015.

[11] MongoDB, Inc, "MongoDB", [Online], https://www.mongodb.org/, diakses tanggal 08 April 2015.

[12] Batra. S, ”AJAX - Asynchronous JavaScript and XML”, Salzburg: Information Technology and System Management, 2003.

BIODATA PENULIS

Muhamad Rohman, biasa dipanggil Romen, lahir di Bojonegoro pada tanggal 02 April 1993. Penulis adalah anak ketiga dari tiga bersaudara. Pendidikan sekolah dasar ditempuh di SDN Ngrejeng, pendidikan menengah di SMPN 1 Purwosari, kemudian dilanjutkan di SMAN 1 Bojonegoro. Tahap sarjana ditempuh di Teknik Informatika Institut Teknologi Sepuluh Nopember (ITS) , dengan spesialisasi bidang minat Komputasi Berbasis Jaringan.

Selama kuliah penulis pernah menjadi asisten beberapa mata kuliah, diantaranya Sistem Digital, Sistem Operasi, dan Jaringan Komputer. Selain itu, penulis juga aktif sebagai Administrator di Laboratorium Arsitektur dan Jaringan Komputer.

Selain aktif dalam kegiatan akademis, penulis juga aktif dalam kegiatan kemahasiswaan, diantaranya aktif di Himpunan Mahasiswa Teknik Computer Informatika, Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi dan Schematics.

Ketertarikan penulis di bidang informatika antara lain pada bidang mikrokontroller dan sekuritas jaringan. Penulis dapat dihubungi melalui surel ke alamat `rohmuhamad@gmail.com`.