APLIKASI *FILE SHARING PEER TO PEER* BERBASIS *WEB* MENGGUNAKAN *WEBRTC*

WEB-BASED PEER TO PEER FILE SHARING APPLICATION USING WEBRTC

Ivan Juan Kara[1], Andi Suprianto[2]
Program Studi Teknik Informatika, Fakultas Sains dan Teknologi Informasi
Institut Sains dan Teknologi Nasional
Jl. Moh Kahfi II, Bhumi Srengseng Indah, Jagakarsa, Jakarta Selatan 12640
Telp. (021) 7874647, Fax. (021) 7866955
ivanjuan222@gmail.com[1], andisuprianto.ta@gmail.com[2]

ABSTRAK

Salah satu kegiatan yang sering dilakukan oleh pengguna komputer adalah *file sharing*. Tidak adanya suatu standar pada sistem operasi populer menjadikan kegiatan ini tidak selalu mudah dilakukan jika ingin mengirim *file* dari dan ke sistem operasi yang berbeda. Instalasi aplikasi pihak ketiga dari *app store* juga bisa merepotkan bagi pengguna awam dan tidak efisien jika hanya sesekali digunakan. Hasil dari penelitian ini adalah aplikasi berbasis *web* yang bisa digunakan untuk melakukan kegiatan *file sharing* secara *peer to peer* dengan menggunakan teknologi *WebRTC* dan kode *QR* sebagai teknik *signalling* alternatif jika *server* sedang *offline* yang dibuat dengan menggunakan *framework Blazor* dan bahasa pemrograman *C#.*
Kata Kunci : *file sharing, peer to peer, WebRTC, kode QR, Blazor*

ABSTRACT

One of the most common thing to do among computer users is file sharing. The lack of standard among popular operating systems makes this task not an easy thing to do if we want to send files from and to different operating systems. Installing apps from third parties will not be easy for novice users and does not seem to be very efficient especially if it is rarely used. The result of this study is a web-based application that can be used to do peer to peer file sharing tasks using WebRTC technology and QR codes as an alternative method of signalling when the server is offline that is made with Blazor framework and C# programming language.
Keywords : *file sharing, peer to peer, WebRTC, QR code, Blazor*

PENDAHULUAN

Salah satu kegiatan yang sering kita lakukan dengan memanfaatkan fasilitas jaringan komputer adalah membagikan *file* dari satu perangkat ke perangkat yang lain. Entah itu membagikan *file* antara perangkat-perangkat milik kita sendiri maupun dari perangkat kita ke perangkat milik orang lain. Biasanya dalam melakukan hal yang sederhana seperti ini, pengguna akan menggunakan protokol *Bluetooth* yang tersedia di hampir semua perangkat. Namun jika pengguna ingin membagikan *file* dengan ukuran besar maka penggunaan *Bluetooth* sangat tidak efisien karena *bandwidth* pada protokol ini sangat kecil dibandingkan dengan teknologi *Ethernet* atau *WiFi*.

Karena tidak ada protokol standar dalam membagikan *file* dengan kedua medium tersebut, maka pengguna harus menggunakan suatu protokol bawaan yang disediakan oleh sistem operasi, misalnya *Windows* dengan *SMB*, dengan konfigurasinya yang tidak terlalu sederhana terutama bagi pengguna awam, atau jika ingin membagikan dengan pengguna yang menggunakan sistem operasi yang berbeda, misalnya dari *Windows* ke *Android* atau *iOS*, maka biasanya pengguna harus memasang suatu program dari pihak ketiga yang dapat berjalan di semua sistem operasi tersebut hanya untuk membagi *file*. Hal ini sangat tidak efisien apabila kegiatan ini tidak sering dilakukan. Masalah lain yang sering penulis pribadi hadapi adalah ketika penulis mau mengirimkan gambar atau video dari perangkat *smartphone* ke komputer penulis untuk dilakukan pengeditan dan pemrosesan lainnya maka penulis biasanya melakukan pengiriman *file* melalui aplikasi *Whatsapp* atau *Telegram* sehingga bisa diterima di

komputer penulis. Cara ini mudah dilakukan karena penulis memiliki kedua aplikasi tersebut di semua perangkat yang penulis miliki. Namun cara ini menjadi sulit dan lama jika *file* yang dikirim berukuran cukup besar karena semua *file* tersebut harus dikirimkan ke *server* milik *Whatsapp* atau *Telegram* sebelum bisa muncul di komputer penulis.

Melihat permasalahan di atas, penulis melalui penelitian ini memutuskan untuk membuat aplikasi berbasis *web* sehingga tidak perlu di-*install* dari *App Store* untuk melakukan *file sharing* secara *P2P* dalam suatu jaringan yang sama yang bisa berjalan di semua sistem operasi yang populer digunakan menggunakan teknologi *WebRTC* dan kode *QR* sebagai teknik alternatif dalam melakukan *WebRTC signalling* ketika *server* dalam kondisi *offline*.

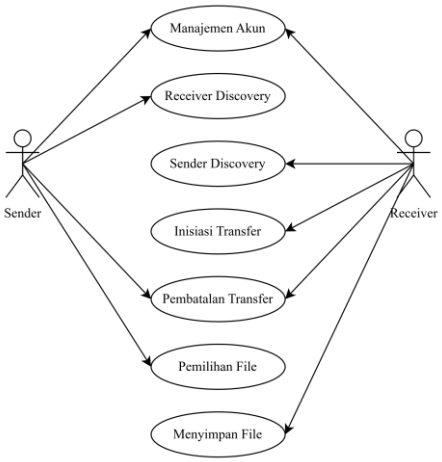
METODOLOGI PENELITIAN

Tahapan Penelitian

Pada penelitian ini, penulis menggunakan metode pengembangan *waterfall* yang sedikit dimodifikasi. Dalam metode yang digunakan terdapat beberapa tahapan yaitu: analisa kebutuhan sistem, perancangan sistem hardware dan software, perancangan aplikasi, uji coba aplikasi. Berikut ini merupakan penjelasan dari masing-masing tahapan tersebut:

Analisa Kebutuhan Sistem

Dalam tahap ini, ditentukan apa saja yang menjadi kebutuhan atau fitur-fitur yang diperlukan supaya sistem ini dapat digunakan dengan baik. Pada umumnya, aplikasi *file sharing* memiliki dua jenis pengguna yaitu pengirim dan penerima. Pengirim harus dapat memilih file yang akan dikirim ke penerima serta penerima harus dapat memilih file apa saja yang mau diambil dari pihak pengirim. Sistem didesain seperti ini demi keamanan dari pihak pengirim. Untuk memudahkan pengguna dalam membedakan antara pengguna satu dengan lainnya, maka dibuat sistem *username*. Baik dari pihak pengirim maupun penerima harus dapat mengubah *username* mereka masing-masing. Selanjutnya, ketika proses transfer *file* sedang berlangsung, baik dari pihak pengirim maupun penerima harus dapat membatalkan proses transfer *file* tersebut. Terakhir, dari pihak penerima harus dapat menyimpan *file* tersebut ke dalam *internal storage* perangkatnya sehingga *file* tersebut nantinya dapat dibuka dan digunakan oleh aplikasi lain. Gambar di bawah ini merupakan *use case diagram* aplikasi berdasarkan penjelasan di atas.



Gambar *Use Case Diagram* Aplikasi

Perancangan Sistem *Hardware* dan *Software*

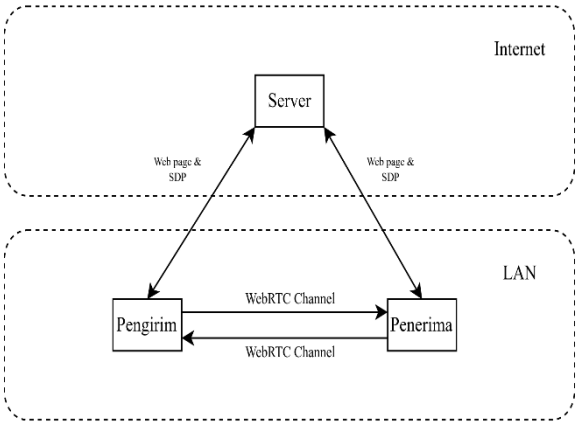
Dalam tahap ini, penulis mendefinisikan kebutuhan sistem yang akan digunakan dalam melakukan penelitian ini. Untuk pembuatan aplikasi, penulis menggunakan perangkat komputer *desktop*. Untuk pengujian, penulis menggunakan perangkat *laptop* dan *smartphone*. Berikut ini merupakan spesifikasi *hardware* yang digunakan dalam pembuatan dan pengujian aplikasi.

Desktop :
CPU 8 core @ 3.4 GHz
Memori 16 GB
Penyimpanan 512 GB

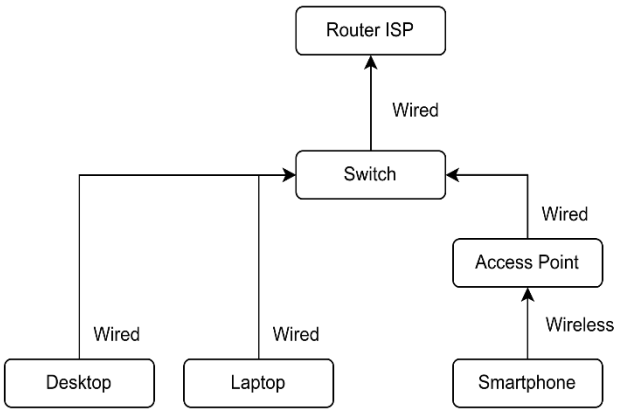
Laptop :
CPU 4 core @ 3.0 GHz
Memori 12 GB
Penyimpanan 250 GB

Smartphone :
Brand Samsung
Model Galaxy M21

Selain perangkat di atas, penulis juga menggunakan satu buah *switch* serta satu buah *access point* sebagai konektivitas antar perangkat yang digunakan. Gambar di bawah ini merupakan diagram koneksi antar perangkat yang digunakan oleh penulis.



Gambar Arsitektur Sistem



Gambar Diagram Koneksi antar Perangkat

Untuk *software* yang digunakan, penulis menggunakan bahasa pemrograman *C#* dan *framework Blazor*. Untuk pengkodingan penulis menggunakan *software Visual Studio Code* dan *dotnet* versi 7. Untuk pengujian aplikasi, penulis menggunakan *browser Microsoft Edge* versi 109 pada semua perangkat yang digunakan.

Perancangan Aplikasi

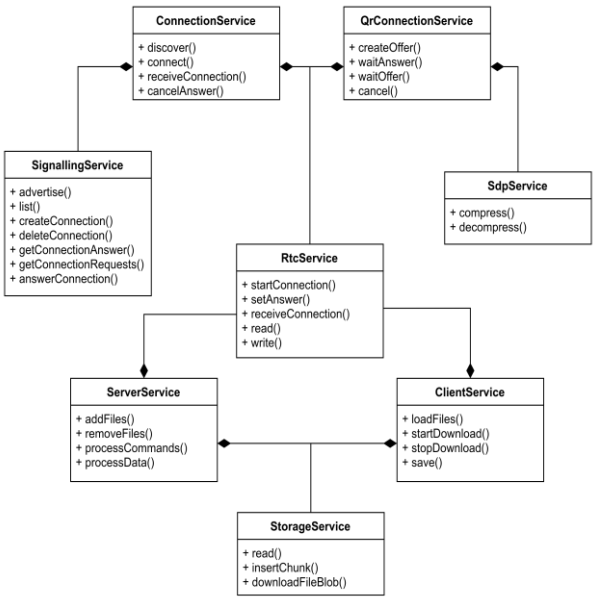
Pada tahapan ini, penulis merancang arsitektur sistem aplikasi, *class diagram*, manajemen akun, teknik *peer discovery* dengan *server* dan kode *QR*, serta protokol *file transfer*.

Arsitektur Sistem

Gambar di bawah ini merupakan arsitektur sistem dari aplikasi yang dibuat. Secara garis besar, aplikasi ini memiliki 2 komponen utama yaitu *server* yang berfungsi sebagai penyedia *static files (html, css, js, wasm)* serta *WebRTC signalling* dan *client* yang merupakan *web browser* seperti *Microsoft Edge, Chrome,* dan *Firefox*. Seperti yang sudah disinggung di awal bahwa aplikasi harus dapat berjalan ketika *server* sedang dalam keadaan *offline*, maka keberadaan *server* di sini bersifat opsional, selama aplikasi web sudah di-*load* setidaknya sekali dan sudah di-*install* sebagai aplikasi *Progressive Web Application (PWA)*. Walaupun *server* sedang dalam keadaan *offline*, aplikasi web ini dapat tetap menjalankan fungsinya karena koneksi antar pengguna terjadi secara langsung *(peer to peer)*.

Class Diagram

Class diagram adalah diagram yang menunjukkan *class* yang ada pada aplikasi dan relasi antar satu *class* dengan *class* yang lain. Gambar di bawah ini merupakan *class diagram* dari aplikasi yang dibuat.



Gambar *Class Diagram* Aplikasi

Berdasarkan gambar *class diagram* di atas, terdapat *class RtcService* yang bertanggung jawab dalam menangani komunikasi dengan *WebRTC data channel* dengan *browser* lain seperti membuat dan menerima koneksi serta mengirim dan menerima data. Selanjutnya ada *class ConnectionService* yang bertanggung jawab untuk melakukan *sender* dan *receiver discovery* serta membuat dan menerima permintaan koneksi. *Class* ini menggunakan *class SignallingService* untuk berkomunikasi dengan *server signalling* dan melakukan pertukaran pesan *Session Description Protocol (SDP)* dengan *client* lain. *Class QrConnectionService* memiliki tanggung jawab yang mirip dengan *class ConnectionService* namun untuk membuat koneksi dengan *client* lain, *class* ini melakukannya dengan menggunakan metode kode *QR*. *Class* tersebut menggunakan class *SdpService* untuk melakukan kompresi dan dekompresi pesan *SDP*. *Class ServerService* dan *ClientService* bertugas untuk menangani *listing file* dan *transfer file* antara *browser* pengirim *(server)* dan penerima *(client)*. Kedua *class* tersebut bergantung kepada *class StorageService* yang berfungsi untuk mengakses *file* dari perangkat pengirim serta melakukan penyimpanan *file* ke perangkat penerima setelah proses transfer selesai.

Manajemen Akun

Ketika aplikasi *web* pertama kali di-*load*, aplikasi akan mengecek pada *IndexedDb* yang ada pada *browser* pengguna apakah sudah ada data *ID* dan *username*. Jika belum ada, maka data tersebut akan di *generate* secara acak lalu disimpan pada *IndexedDb browser*. Untuk proses *load* selanjutnya, maka tidak perlu lagi melakukan *generate* terhadap data tersebut. *ID* dan *username* digunakan untuk memudahkan pengguna dalam membedakan antara pengguna yang lainnya pada saat proses *peer discovery*. Seperti yang sudah disinggung pada bagian sebelumnya, *username* dapat diganti oleh pengguna sesuai dengan keinginannya. Perubahan *username* tersebut juga disimpan ke *IndexedDb* pada *browser*.

Peer Discovery* dengan *Server

Supaya setiap *client* bisa saling terhubung maka diperlukan suatu teknik *signalling* untuk saling bertukar pesan *Session Description Protocol (SDP)*. Salah satu teknik yang paling umum digunakan yaitu dengan menggunakan suatu *server* terpusat yang bisa dijangkau oleh seluruh *client*. Secara sederhana, tugas dari *server* ini hanyalah meneruskan pesan *SDP* dari satu *client* ke *client* yang lain.

Misalnya *client* A ingin membuat *channel WebRTC* ke *client* B. *Client* A pertama-tama membuat pesan *SDP offer* kemudian pesan ini dikirimkan ke *server signalling*. *Server signalling* menerima pesan tersebut kemudian memberitahu *client* B bahwa ada *client* yang ingin membuat koneksi dengannya dan mengirimkan pesan *SDP offer* dari *client* A ke *client* B. *Client* B kemudian membuat pesan *SDP answer* berdasarkan pesan *SDP offer* dari *client* A tersebut. Selanjutnya *client* B mengirimkan pesan *answer* tersebut kembali ke *server* yang kemudian oleh *server* tersebut diteruskan kembali ke *client* A. Sampai di sini, kedua *client* tersebut sudah bisa membuat *channel WebRTC* dan berkomunikasi satu sama lain tanpa melalui *server* lagi.

Peer Discovery* dengan Kode *QR

Signalling menggunakan *server* merupakan cara yang terbaik dan termudah bagi pengguna. Namun cara ini tidak akan bekerja jika pengguna sedang tidak terkoneksi ke *internet* atau jika server *signalling* sedang *down* atau *maintenance*. Karena inti dari koneksi *WebRTC* adalah pertukaran pesan *SDP* antara dua *web browser* maka secara teknis kita bisa menggunakan medium apapun untuk bertukar pesan tersebut seperti melalui *instant messaging*, *bluetooth*, menulis manual, atau dengan menggunakan kode *QR*. Dalam aplikasi ini akan digunakan metode kode *QR* sebagai medium alternatif untuk bertukar pesan *SDP*.

Langkah-langkahnya adalah *client* A membuat pesan *SDP offer*, kemudian *client* A membuat kode *QR* dari pesan tersebut. Lalu *client* B melakukan *scan* terhadap kode *QR* dari *client* A tadi. Setelah itu *client* B membuat pesan *SDP answer* berdasarkan pesan *SDP offer* dari *client* A, lalu membuat kode *QR* dari pesan *SDP answer* tersebut. Selanjutnya *client* A melakukan *scan* terhadap kode *QR* dari *client* B dan pertukaran pesan *SDP* selesai. Sampai di sini kedua *client* bisa melakukan koneksi dengan menggunakan *channel WebRTC* satu sama lain tanpa memerlukan bantuan dari *server signalling* sama sekali.

Selanjutnya untuk bisa saling mengetahui *username* masing-masing, aplikasi pengirim mengirimkan informasi *ID* dan *username* yang digunakannya kepada aplikasi penerima. Setelah itu, aplikasi penerima menampilkan *username* pengirim tersebut pada halaman *Sender Discovery,* lalu membalas pesan dari aplikasi pengirim tadi dengan informasi *ID* dan *username* yang digunakan oleh penerima. Aplikasi pengirim menerima pesan

balasan tersebut lalu menampilkan *username* penerima pada halaman *Receiver Discovery*.

Salah satu kesulitan yang dihadapi dalam menerapkan teknik *signalling* dengan kode *QR* yaitu ukuran dari pesan *SDP* yang terlalu besar untuk ditampung dalam satu kode *QR*. Kode *QR* hanya dapat menampung informasi sebesar kurang dari 200 karakter. Lebih dari itu, maka *scanner* kode *QR* akan mengalami kesulitan dalam melakukan *scan* kode tersebut atau bahkan tidak bisa di-*scan* sama sekali. Biasanya ukuran pesan *SDP* adalah 500 karakter lebih sehingga melakukan kompresi terhadap pesan *SDP* menjadi suatu keharusan. Berikut ini merupakan contoh pesan SDP sebelum dilakukan kompresi:

```
a=ice-ufrag:hqkw

a=ice-options:trickle
a=fingerprint:sha-256
06:54:9D:FE:0E:C7:0A:26:61:31:34:CA:BF:30
:FF:15:7F:DC:48:1C:7D:91:4B:9B:72:59:F9:7
E:DA:AE:BD:9D
a=setup:actpass
a=mid:0
a=sctp-port:5000
a=max-message-size:262144
```

Mengutip dari *webrtcHacks*, pesan *SDP* memiliki beberapa komponen penting berikut ini :

- `ice-ufrag`
- `ice-pwd`
- *SHA-256 DTLS fingerprint*
- *ICE candidates*

`ice-ufrag` dan `ice-pwd` bisa diibaratkan sebagai *username* dan *password*. *SHA-256 DTLS fingerprint* dapat dikompresi dengan menghilangkan `a=fingerprint:sha-256` lalu mengkonversikan ke dalam bentuk *base64*. Untuk informasi *ICE candidates*, kita bisa mengambil bagian `4171069048 1 udp 2122260223 192.168.20.93 53290 typ host generation 0 network-id 2 network-cost 10` dan menghilangkan sisanya. Berikut ini merupakan penjelasan dari baris *ICE candidate* :

1. `4171069048` merupakan *foundation*. Dapat dikompresi ke bentuk *base64*.

```
v=0
o=- 712533869939805848 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE 0
a=extmap-allow-mixed
a=msid-semantic: WMS
m=application 9 UDP/DTLS/SCTP webrtc-
datachannel
c=IN IP4 0.0.0.0
a=candidate:4171069048 1 udp 2122260223
192.168.20.93 53290 typ host generation 0
network-id 2 network-cost 10
a=candidate:3300162712 1 udp 2122194687
10.33.121.97 60553 typ host generation 0
network-id 1 network-cost 900
a=ice-pwd:GC/S47JXwoFx2B0qjvwPJqKg
```

2. `1` merupakan *component* yang berarti hanya menggunakan *data channel.* Bisa diabaikan karena selalu statis.
3. `udp` merupakan protokol *transport* yang digunakan. Bisa diabaikan karena kita hanya menggunakan *UDP*.
4. `2122260223` merupakan priority. Bisa dikompresi ke bentuk *base64*.
5. `192.168.20.93 53290` merupakan IP address dan port.
6. `typ host generation 0` selalu statis dan bisa diabaikan.
7. `network-id 2 network-cost 10` cukup diambil angka 2 dan 10.

Hal yang sama juga diterapkan terhadap *ICE candidate* kedua. Informasi yang tidak disebutkan atau diabaikan di atas bisa di-*hardcode* di dalam aplikasi sehingga tidak perlu disertakan dalam kode *QR*. Berikut merupakan contoh pesan *SDP* hasil kompresi dari pesan *SDP* sebelumnya :

```
hqkw;GC/S47JXwoFx2B0qjvwPJqKg;BlSd/g7HCiZ
hMTTKvzD/FX/cSBx9kUubcln5ftquvZ0=;O;eHad+
A== /x5/fg== 192.168.20.93 53290 10
2;mHy0xA== /x5+fg== 10.33.121.97 60553 900
1
```

Protokol *File Transfer*

Setelah kedua *client* saling terkoneksi maka pengirim bisa memilih *file* untuk bisa di-*download* oleh penerima. Ketika *file* sudah dipilih oleh pengirim, maka penerima dapat melakukan *listing* informasi *file* yang sudah dipilih oleh pengirim

seperti nama dan ukuran *file*. Penerima pun dapat memilih *file* mana saja yang mau di-*download*.

Ketika tombol *download* diklik oleh penerima, maka aplikasi penerima akan mengirimkan *request* untuk *download file* dengan *offset* ke-0 dengan ukuran *n* kepada aplikasi pengirim. Contohnya, jika *file* berukuran 10 MB dan jika *n* = 1 MB, maka proses permintaan ini akan terjadi sebanyak 10 kali. Hal ini dilakukan supaya dari pihak pengirim bisa melakukan pembatalan terhadap transfer *file* sebelum proses transfer *file* selesai. Di setiap proses *request* akan ditampilkan *progress* transfer *file* di aplikasi penerima sehingga pengguna bisa tahu sejauh mana proses transfer tersebut berjalan. Setelah proses transfer selesai, maka penerima dapat menekan tombol *save* pada *file* tersebut untuk memilih lokasi penyimpanan *file* supaya bisa digunakan oleh aplikasi lainnya jika diperlukan.

HASIL DAN PEMBAHASAN

Tampilan Halaman Utama

Halaman utama merupakan halaman yang pertama kali ditampilkan kepada pengguna ketika aplikasi pertama kali di-*load*. Gambar di bawah ini merupakan tampilan utama aplikasi yang dibuat pada perangkat *desktop*.

Gambar Tampilan Utama Aplikasi

Gambar Tampilan *Settings*

Tampilan *Receiver Discovery*

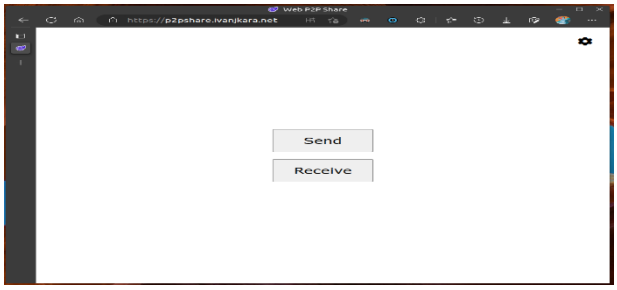
Halaman *Receiver Discovery* merupakan tempat di mana pengirim dapat melihat daftar calon penerima *file* yang berhasil terdeteksi oleh aplikasi, baik melalui *server signalling* atau *QR signalling*. Gambar di bawah ini merupakan tampilan dari halaman *Receiver Discovery*.

Uji Coba Aplikasi

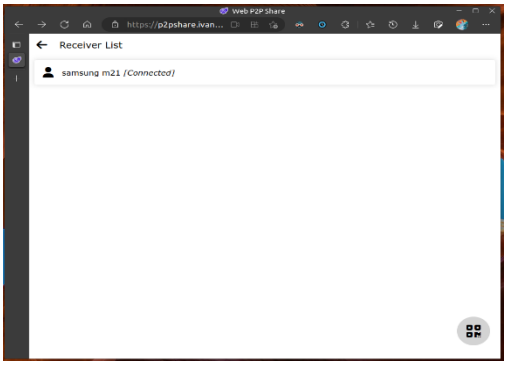
Uji coba aplikasi merupakan salah satu tahapan penting dalam pembuatan aplikasi. Kegiatan ini bertujuan untuk menemukan kesalahan atau *bugs* yang ada pada aplikasi yang dibuat sebelum aplikasi digunakan oleh *end user*. Metode uji coba aplikasi yang penulis pakai untuk aplikasi ini yaitu metode pengujian *blackbox*. Menurut (Iskandaria, 2012), pengujian *blackbox* adalah metode pengujian perangkat lunak yang berfokus pada sisi fungsionalitas, khususnya pada *input* dan *output* aplikasi, apakah sudah sesuai dengan yang diharapkan atau belum.

Tampilan *Settings*

Halaman *settings* dapat diakses melalui tombol gerigi yang ada pada sisi kanan atas halaman utama. Pada halaman ini, pengguna bisa melihat *username* yang digunakan serta melakukan perubahan terhadap *username* tersebut. Gambar di bawah ini merupakan tampilan dari halaman *settings*.



Gambar Tampilan *Sender Discovery*



Gambar Tampilan *Receiver Discovery*

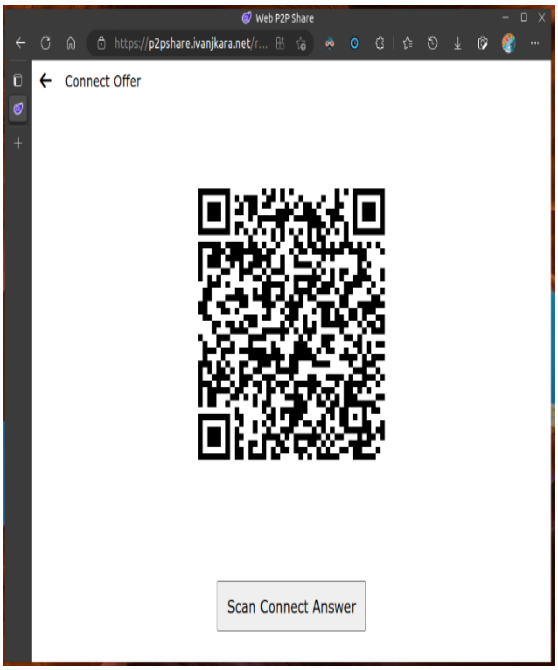
Tampilan *Sender Discovery*

Berbanding terbalik dengan Receiver Discovery, Sender Discovery merupakan tempat di mana penerima *file* dapat melihat daftar pengirim yang terdeteksi oleh aplikasi. Gambar di bawah ini merupakan tampilan dari *Sender Discovery*.

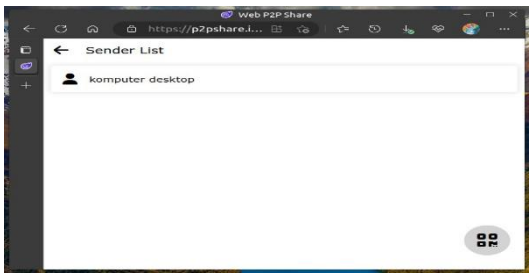
Tampilan *QR Signalling*

Jika *server signalling* sedang dalam keadaan *offline*, maka antar pengguna tidak akan bisa saling menemukan satu sama lainnya. Metode lain untuk menghubungkan antara pengguna satu dengan halaman *Connect Answer* yang menampilkan kode *QR* yang berisi *SDP Answer*. Pengirim kemudian melakukan scan terhadap kode *QR* tersebut. Setelah kedua pengguna saling bertukar kode *QR*, maka perangkat kedua pengguna tersebut sudah saling terhubung secara langsung menggunakan *WebRTC*. Gambar di bawah ini merupakan tampilan dari halaman *Connect Offer* pada sisi pengirim.

Gambar di bawah ini merupakan tampilan dari halaman *Scan Connect Answer* dari sisi pengirim.

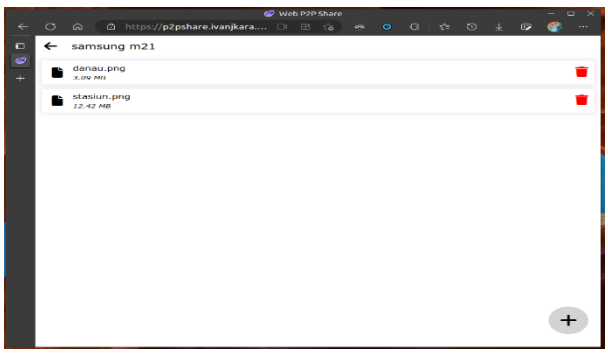


lainnya adalah dengan menggunakan kode *QR* sebagai media pertukaran pesan *SDP*. Caranya adalah dari sisi pengirim, pada halaman *Receiver Discovery*, menekan tombol kode *QR* yang ada pada sisi kanan bawah. Kemudian pada sisi pengirim akan ditampilkan kode *QR* yang berisi *SDP Offer*. Kode *QR* ini bisa di-*scan* oleh pengirim pada halaman *Scan Connect Offer* yang bisa dibuka dari halaman *Sender Discovery*. Setelah proses *scan* kode *QR* berhasil, pengirim menekan tombol *Scan Connect Answer*. Pada sisi penerima akan otomatis berubah ke



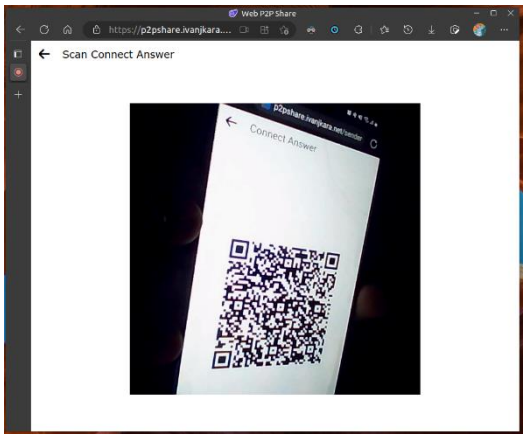
Gambar Tampilan *Connect Offer* Pengirim

Gambar di bawah ini merupakan tampilan dari halaman *Scan Connect Offer* di sisi penerima.



Gambar Tampilan *Scan Connect Offer* Penerima

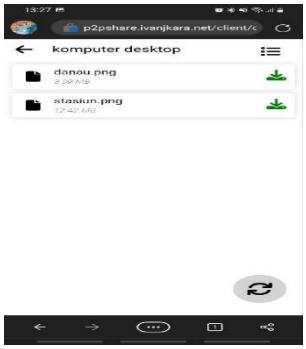
Gambar di bawah ini merupakan tampilan dari halaman *Connect Answer* dari sisi penerima.



Gambar Tampilan *Scan Connect Answer* Penerima

Tampilan *File List* Sisi Pengirim

Setelah pada halaman *Receiver Discovery* pengirim memilih penerima, maka akan dialihkan ke halaman *File List*. Pengirim bisa memilih *file* apa saja yang akan dikirimkan ke penerima. Gambar di bawah ini merupakan tampilan dari halaman *File List* pada sisi pengirim.



Gambar Tampilan *File List* sisi Pengirim

Tampilan *File List* Sisi Penerima

Setelah penerima memilih pengirim pada halaman *Sender Discovery*, maka akan dialihkan ke halaman *File List*. Di sini, penerima dapat memilih *file* yang sudah dipilih oleh pengirim dari perangkatnya. Pada halaman ini juga, penerima dapat melakukan inisiasi proses *download* terhadap *file-file* tersebut. Gambar di bawah ini merupakan tampilan halaman *File List* pada sisi penerima.

Gambar Tampilan *File List* sisi Penerima

Tampilan *Progress Transfer File*

Untuk dapat melihat sampai sejauh mana progress pengiriman *file* pada sisi penerima, penerima dapat menekan tombol yang ada pada sisi kanan atas di halaman *File List*. Tombol tersebut akan menunjukkan halaman *Progress Transfer File* yang menampilkan *file* apa saja yang sedang di-*download*, yang sudah di-*download*, maupun yang sedang menunggu antrian proses *download*. Gambar di bawah ini merupakan tampilan dari halaman *Progress Transfer File*.

Gambar Tampilan *Progress Transfer File*

Tabel Hasil Pengujian Aplikasi

Pengujian	Hasil
Pengirim dan penerima dapat melakukan penggantian *username* pada halaman *settings*	Berhasil
Pengirim dapat melihat daftar penerima	Berhasil
Pengirim dapat membuat koneksi ke penerima	Berhasil
Penerima dapat menerima permintaan koneksi dari pengirim	Berhasil
Listing pengguna lain saat *server signalling* dalam kondisi mati	Tidak Berhasil
Pengirim dan penerima dapat saling terkoneksi dengan menggunakan kode *QR* ketika *server signalling* mati	Berhasil
Proses *transfer file* berjalan dengan baik dan *file* diterima dengan baik pada sisi penerima	Berhasil
Pengirim dapat melakukan pembatalan transfer *file* saat proses masih sedang berlangsung	Berhasil

Hasil Uji Coba Aplikasi

Tabel di bawah ini merupakan rekapitulasi hasil uji coba aplikasi dengan menggunakan metode *blackbox*.

PENUTUP

Simpulan

1. Telah dirancang aplikasi berbasis *web* untuk melakukan kegiatan *file sharing* secara *peer to peer* dalam suatu jaringan *LAN* yang sama dengan memanfaatkan teknologi *WebRTC* sebagai media komunikasi antar *client*.
2. Aplikasi dapat di-*install* sebagai *Progressive Web Application (PWA)* sehingga jika nantinya *server* sedang *offline* maka aplikasi tetap bisa dibuka di perangkat pengguna.

3. Metode *WebRTC signalling* menggunakan kode *QR* digunakan untuk saling bertukar pesan *Session Description Protocol (SDP)* ketika *server* dalam kondisi *offline* supaya masing-masing *client* tetap bisa terhubung satu sama lain.
4. Aplikasi berhasil diuji coba dengan menggunakan perangkat *desktop*, *laptop*, dan *smartphone* yang saling terhubung dalam satu jaringan *LAN* yang sama dengan mengirimkan *file* gambar berjenis *.png* dari perangkat *desktop* ke perangkat *laptop* dan *smartphone.* Hasilnya *file* dapat diterima dengan baik pada kedua perangkat tersebut.
5. Kelebihan dari aplikasi ini yaitu dapat berjalan di hampir semua sistem operasi dan tidak perlu di-*install* dari *App Store* karena aplikasi ini berjalan pada *platform web*. Selain itu, aplikasi ini bisa tetap menjalankan fungsinya dengan baik walaupun *server* dalam kondisi *offline*.
6. Kekurangan dari aplikasi ini yaitu tidak bisa melakukan proses transfer *file* ketika layar *smartphone* dalam kondisi mati. Ini dikarenakan *web browser* pada *smartphone* menerapkan teknik *power saving* dengan menghentikan semua eksekusi kode pada aplikasi *web* ketika layar *smartphone* mati setelah beberapa menit. Hal ini tidak berlaku pada perangkat *desktop* dan *laptop*.

Saran

1. Untuk penelitian selanjutnya, perlu dikembangkan metode komunikasi antar perangkat menggunakan *WebRTC* yang tetap berjalan saat layar *smartphone* mati.
2. Perlu dikembangkan metode untuk mengecilkan ukuran *file* aplikasi *web* sehingga aplikasi dapat diproses lebih cepat pada saat *loading* pertama.

DAFTAR PUSTAKA

Trilogy-LTE. (2014). *How WebRTC Is Revolutionizing Telephony.* Diakses pada 10 Desember 2022, dari https://blogs.trilogy-lte.com/ post/77427158750/how-webrtc-is-revolutionizing-telephony

Hancke, Philipp. (2015). *The Minimum Viable SDP*. Diakses pada 10 Desember 2022, dari https://webrtchacks.com/the-minimum-viable-sdp

Iskandaria. (2013). *Contoh Pengujian Black Box*. Diakses pada 10 Desember 2022, dari https://web.archive.org/web/20130528111656/ https://kafegue.com/Contoh-Pengujian-Black-Box-Testing