DOKUMEN KARYA ILMIAH
SYNCFOLDER SEBAGAI ALAT SINKRONISASI FILE DAN FOLDER DENGAN MENGGUNAKAN PROTOKOL MANDIRI

Periode Penilaian Semester I Tahun 2019

Oleh:

Budi Satrio
NIP 198503082010121009
Sekretariat Jenderal

Tim Penilai Instansi Pusat

Jabatan Fungsional Pranata Komputer

Kementerian Keuangan R.I.

Jakarta, April 2019

Dokumen Karya Ilmiah SyncFolder sebagai Alat Sinkronisasi File dan Folder dengan menggunakan Protokol Mandiri
Tanggal : 22 April 2019
Disusun oleh :

NAMA/ NIP	JABATAN	TANDATANGAN
Budi Satrio/ 19850308 201012 1 009	Pranata Komputer Bidang Pengembangan Sistem Informasi	

Penanggung Jawab:

Kepala Bidang Pengembangan
Sistem Informasi,

Yusuf Nurrohman
NIP 19750802 199502 1 001

Diterbitkan oleh:
Pusat Sistem Informasi dan Teknologi Keuangan
Jalan Lapangan Banteng Timur Nomor 2–4, Jakarta 10710
Situs Web http://pusintek.kemenkeu.go.id
Telephone +62.21.3449230 external 4100
Faksimile +62.21.3519655
E-mail servicedesk@kemenkeu.go.id

LEMBAR PENGESAHAN

Dengan menandatangani lembar ini semua pihak menyatakan telah membaca, memahami, dan menyetujui isi Karya Ilmiah SyncFolder sebagai Alat Sinkronisasi File dan Folder dengan menggunakan Protokol Mandiri.

Diperiksa Oleh:

Nama/ NIP	Jabatan	Tandatangan	Tanggal
Yusuf Nurrohman/ 19750802 199502 1 001	Kepala Bidang Pengembangan Sistem Informasi		23/4/19

Disetujui Oleh:

Nama/ NIP	Jabatan	Tandatangan	Tanggal
Herry Siswanto/ 19710322 199603 1 002	Kepala Pusat Sistem Informasi dan Teknologi Keuangan		23/4/19

DAFTAR ISI

DAFTAR GAMBAR

ABSTRAKSI

SyncFolder merupakan sebuah tool yang dibangun untuk keperluan sinkronisasi *file* dan folder antara server-server yang terletak dalam *Data Center* dan *Disaster Recovery Center* Kementerian Keuangan. SyncFolder menggunakan protokol mandiri yang pembangunannya dibuat berdasarkan prinsip komunikasi dua arah *secara half duplex* dengan menggunakan teknologi *transfer protocol*.

Seiring dengan digunakannya SyncFolder di *Data Center* dan *Disaster Recovery Center* Kementerian Keuangan, terdapat request untuk menambahkan fitur *monitoring* dan administrasi. Fitur monitoring ditambahkan dengan menggunakan teknologi *mini web server* dengan protokol http, sedangkan fitur administrasi ditambahkan dengan teknologi client server dengan bantuan tool SyncFolderAdmin sebagai *aggregator* dan *command center* bagi aplikasi SyncFolder.

Terdapat beberapa keunggulan aplikasi SyncFolder serta tantangannya jika dibandingkan dengan aplikasi berbayar lain, diantaranya yaitu: Kemampuan untuk melakukan sinkronisasi berdasarkan *delta* (mencari selisih perbedaan *file* dan intelegensi untuk menentukan *file* mana yang akan ditimpa atau tidak), Kemampuan untuk berjalan menggunakan *communication port* yang bebas, serta Kemampuan untuk transfer data secara cepat karena menggunakan protokol mandiri. Tantangannya yaitu produk ini dibuat oleh pegawai Kementerian Keuangan secara pribadi, sehingga diperlukan peran aktif pengguna untuk melaporkan setiap *bug* yang muncul pada aplikasi.

Keywords: **Sinkronisasi, Sync File dan Folder, *Half duplex, Protocol, Transfer Protocol, HTTP, Monitoring, Web Server, Delta, Communication Port, Fast Transfer***

Budi Satrio

NIP. 198503082010121009

BAB I. PENDAHULUAN

Dokumen karya ilmiah ini dibuat dengan tujuan memperkenalkan SyncFolder yang sekarang digunakan oleh pengelola aplikasi di Data Center dan Disaster Recovery Center Kementerian Keuangan.

A. Latar Belakang

Berawal dari kebutuhan sinkronisasi data antara portal Kementerian Keuangan yang mempunyai topologi multiple server, penulis diminta untuk membantu proses sinkronisasi tersebut. Diawali dari menggunakan *command* prompt untuk pengiriman *file* dan folder secara berulang menggunakan scheduler, kemudian berkembang menjadi SyncFolder namun menggunakan Windows Sharing Folder, sampai kepada menggunakan protokol yang dibuat sendiri.

Semakin lama penggunaan SyncFolder semakin banyak sehingga dibutuhkan fitur untuk monitoring yang mudah diakses dari mana saja. Oleh karena itu perkembangan selanjutnya adalah penambahan fitur monitoring web untuk setiap aplikasi SyncFolder. Ternyata monitoring saja tidak cukup, sehingga dibutuhkan juga fitur pengelolaannya termasuk start, stop, dan get status. Saat karya ilmiah ini ditulis, fitur pengelolaan ini telah diimplementasikan pada SyncFolder versi terbaru

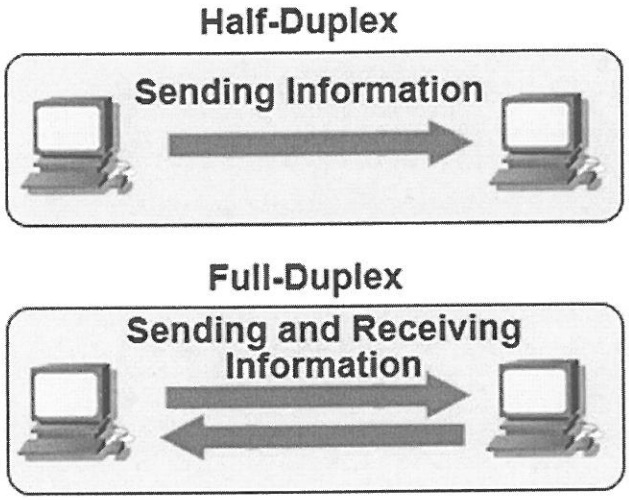
dan untuk manajemennya menggunakan tool lain lagi bernama SyncFolderAdmin.

B. Ruang Lingkup

Karya ilmiah ini hanya berfokus pada aplikasi SyncFolder beserta teknis dalamnya berupa protokol, topologi, tampilan antarmuka, alur instalasi dan penggunaan aplikasi, serta keunggulan dan tantangannya. Karya ilmiah ini tidak membahas detail mengenai aplikasi SyncFolderAdmin, maupun aplikasi lain yang digunakan sebagai perbandingan keunggulan aplikasi SyncFolder. Permasalahan yang muncul pada penggunaan aplikasi SyncFolder ini juga bukan merupakan ruang lingkup karya ilmiah ini.

BAB II. POKOK BAHASAN

Pada bagian ini akan dijelaskan bagaimana prosedur teknis sinkronisasi dengan protokol yang dibuat secara mandiri. Mekanisme protokol mandiri mengikuti kaidah komunikasi data secara *half duplex* yang meliputi *Handshake, Checking*, serta *Sending* dan *Receiving*[1]. *Duplex Transmission System* adalah metode komunikasi dua arah antar perangkat telekomunikasi dimana didalamnya terbagi menjadi dua kubu besar, yaitu *full duplex* dan *half duplex*.



Gambar 1. Perbedaan *Half duplex* dan *Full duplex*

Full duplex adalah metode komunikasi dua arah dimana setiap pengirim dan menerima dapat mengirimkan dan menerima pesan dalam

waktu yang bersamaan secara bersama-sama. Contoh dari komunikasi *full duplex* adalah saluran telepon, dimana penelpon dan penerima dapat saling berbicara dan saling mendengar secara bersama-sama. Kekurangan dari metode ini adalah umumnya menggunakan dua jalur komunikasi sehingga investasi menjadi lebih mahal, namun kelebihannya adalah *real time*.

Half duplex adalah metode komunikasi dua arah dimana pengirim dan penerima dapat mengirimkan dan menerima pesan secara bergantian. Contoh dari komunikasi *half duplex* ini adalah *walkie-talkie*, dimana untuk dapat berkomunikasi, pengirim pesan harus menekan tombol bicara dimana pada saat itu si penerima tidak akan bisa mengirim pesan, namun bisa mendengarkan. Kelebihan dari metode ini adalah transmisi komunikasi yang cepat, dan biaya yang cenderung murah dibandingkan *full duplex*, namun kelemahannya adalah komunikasi tidak real time, tetapi dapat diakali dengan menggunakan perpindahan waktu pengiriman pesan yang cepat..

Dengan melihat kelebihan dan kekurangan antara dua metode tersebut, penulis memilih untuk menggunakan metode *half duplex* dengan membuat protokol komunikasi yang mudah dan tidak serumit komunikasi standar pada umumnya. Hal ini diperkuat juga dengan alasan keamanan. Jika protokol yang dibuat secara standar maka setiap orang akan dengan mudah mengetahui dan mencari kelemahan keamanan dari protokol tersebut,

sebagai contoh adalah kasus *Heartbleed*, yaitu protokol HTTP Secure versi lama yang ternyata mudah dibobol oleh hacker[2].

A. *Communication* dan *Command key*

Pada prinsipnya, setiap kali transmisi dilakukan oleh *sender*, *receiver* harus melakukan proses yang diminta oleh *sender* (*acknowledge*). Untuk memudahkan proses komunikasi dengan metode *half duplex*, pembuatan protokol ini harus memperhatikan jenis-jenis *command key* dan data-data yang akan dikirimkan. Oleh karena itu, *command key* pada protokol mandiri ini dibuat dengan menggunakan kosakata dalam bahasa inggris yang sifatnya adalah *compressed*, gunanya agar mudah dibaca namun tidak merepotkan mesin untuk menginterpretasikannya. Sebagai contoh adalah TESTCON untuk melakukan pengujian koneksi (*test connection*) dan DELDIR untuk menghapus direktori (*delete directory*).

Terdapat 17 jenis *command key* yang digunakan dalam protokol ini, yaitu:

1. START<EOF>

Command ini merupakan awal perintah dari *sender* kepada *receiver* untuk bersiap menerima kiriman. *Receiver* akan mengirimkan balasan kepada *sender* berupa alamat tujuan directory di *receiver* yang akan dilakukan sinkronisasi.

2. **TESTCON<EOF>**

Command ini merupakan gabungan dari kata *Test Connection*, yaitu perintah dari *sender* untuk melakukan pengujian koneksi antara *sender* dan *receiver*. *Receiver* tidak mengirimkan balasan kepada *sender*, namun menampilkan informasi *sender* pada log *receiver* bahwa test koneksi berhasil.

3. **GETJOBS<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk mengirimkan pekerjaan-pekerjaan apa saja yang belum diselesaikan. Pekerjaan dalam hal ini adalah sisa sinkronisasi yang belum berhasil dijalankan sebelumnya di *receiver*. Hal ini bisa terjadi karena beberapa hal, seperti invalid permission untuk menuliskan *file*, putusnya jaringan saat transfer data sedang dilakukan, atau penambahan *file* di *receiver* yang perlu untuk dicek oleh *sender*.

Receiver akan menjawab *sender* dengan isi pesan yaitu detail pekerjaan yang belum diselesaikan dalam bentuk format *Java Script Notation Object* (JSON). Format ini dipilih oleh penulis karena kehandalannya dalam melakukan serialize object menjadi sebuah string. Format ini sering ditemukan dalam aplikasi-aplikasi web.

4. **NUMJOBS[number]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk menyimpan angka jumlah pekerjaan sinkronisasi yang belum dijalankan. [number] berisi angka jumlah pekerjaan tersebut, sehingga jika terdapat dua pekerjaan, maka *command* akan menjadi NUMJOBS2. Pekerjaan dalam hal ini adalah proses sinkronisasi yang dipecah per *file* atau folder yang dikirimkan. Sebagai contoh ketika folder *sender* kita hanya mempunyai dua *file*, A dan B, dan akan disinkronkan ke *receiver*, maka jumlah pekerjaan tersebut adalah dua, yaitu 1. sinkronkan *file* A, dan 2. Sinkronkan *file* B.

Maksud dari *command* ini sebetulnya adalah untuk menyiapkan progress bar yang ditampilkan di sisi *receiver*, karena *receiver* tidak akan mengetahui berapa total dari sejumlah sinkronisasi yang sudah berjalan. Dengan adanya *command* ini, *receiver* dapat membuat perkiraan seberapa persen dari total pekerjaan dimana sinkronisasi ini berjalan.

5. **WATCH[destDIr]<SOURCE>[sourceDir]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk memperhatikan lokasi direktori pada *receiver* sesuai dengan lokasi yang dikirimkan oleh *sender*. Hal ini

berguna pada mode delta, yaitu melakukan sinkronisasi kembali untuk *file* atau folder yang belum pernah atau berbeda pada sinkronisasi sebelumnya.

6. **ENDPROG<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk mengakhiri sesi sinkronisasi. *Receiver* akan membalas dengan status ok lalu kemudian melakukan penutupan koneksi.

7. **DELJOB[sourcePath]<EOF>**

Command ini merupakan perintah dari *sender* untuk menghapus job sinkronisasi *file* berdasarkan lokasi yang dikirimkan oleh *sender*. Hal ini berguna agar tidak terjadi duplikasi sinkronisasi *file*.

8. **CHECKDIR[destPath]<EOF>**

Command ini merupakan perintah dari *sender* untuk mengecek apakah direktori sesuai dengan lokasi yang dikirimkan *sender* ada pada *receiver*. *Receiver* akan membalas dengan status direktori tersebut eksis attau tidak.

9. **DELDIR[destPath]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk menghapus direktori beserta *file* di dalamnya

sesuai dengan lokasi yang dikirimkan oleh *sender*. *Receiver* akan membalas dengan status penghapusan direktori apakah berhasil atau tidak.

10. **CREATEDIR[destPath]<EOF>**

Command ini merupakan perintah dari *sender* untuk membuat direktori pada *receiver* sesuai dengan lokasi yang dikirimkan oleh *sender*. *Receiver* akan membalas dengan status bahwa pembuatan direktori sukses atau tidak.

11. **CHECKFILE[destPath]<EOF>**

Command ini merupakan perintah dari *sender* untuk melakukan pengecekan apakah *file* ada pada *receiver* sesuai dengan lokasi yang dikirimkan oleh *sender*. *Receiver* akan membalas dengan *status file* tersebut apakah ada atau tidak.

12. **DELFILE[destPath]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk menghapus *file* sesuai dengan lokasi *file* yang dikirimkan oleh *sender*. *Receiver* akan membalas kepada *sender* dengan status delete apakah berhasil atau tidak.

13. **CHECKFLDIR[destPath]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk mengecek lokasi direktori pada *receiver* apakah

ada dan eksis. *Receiver* akan membalas dengan status bahwa direktori telah ada atau tidak.

14. **CHECKFLDATA[destPath]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk mengecek apakah *file* yang berada pada lokasi yang dikirimkan oleh *sender*, ada dan eksis pada *receiver*. *Receiver* akan mengirimkan *status file* tersebut pada *sender*. *Sender* kemudian akan melakukan pengecekan fike tersebut. Hal ini diperlukan agar tidak terjadi pengiriman ulang *file* yang sudah ada dan sama dengan yang di *sender*.

15. **CREATEFL[savePath]<DATE>[dateModified]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk membuat *file* kosong di *receiver* dengan lokasi dan tanggal modifikasi sesuai dengan yang dikirimkan oleh *sender*. *Receiver* akan mengirimkan balik status pembuatan *file* apakah sukses atau tidak.

16. **SENDFILE[savePath]<SIZE>[sizeFile]<DATE>[dateModified]<EOF>**

Command ini merupakan perintah dari *sender* untuk mengirimkan *file* kepada *receiver* dimana didalamnya berisi lokasi penulisan *file* pada *receiver*, ukuran *file*, beserta tanggal

modifikasi *file* tersebut. *Sender* akan mengirimkan byte per byte dari *file* melalui jaringan yang sudah terhubung sebelumnya. Selama *receiver* masih dalam mode penerimaan *file*, semua byte yang dikirimkan oleh *sender* akan ditulis ke dalam disk drive di *receiver*. *Receiver* perlu mengetahui ukuran *file* dan date modified nya untum memastikan *file* yang dikirimkan adalah sama. *Receiver* akan mengirimkan balik status penulisan *file* apakah sukses atau tidak.

17. **ENDSENDFL[savePath]<EOF>**

Command ini merupakan perintah dari *sender* kepada *receiver* untuk mengakhiri mode transfer *file*. *Receiver* akan menutup penulisan *file* pada direktori dengan end of *file*, lalu mengembalikan lokasi *file* beserta direktorinya kepada *sender*. Hal ini diperlukan karena *receiver* akan menganggap semua byte yang masuk adalah bagian dari *file* dan bukan berupa *command*.

BAB III. ALGORITMA UTAMA

Pada bagian ini, akan dijelaskan mengenai algoritma utama aplikasi, yang terdiri dari *sending, receiving, monitoring*, dan *admin*.

A. *Sending*

Sending merupakan proses pengiriman *file* atau folder dari *sender* menuju *receiver* dengan menggunakan protokol mandiri. Berikut adalah algoritma dan penjelasannya.

```
private void Sending
{
 inisiasiJobs()
 while (tidakAdaInterrupt)
 {
  if (menggunakanPropietaryProtocol)
  {
   nyalakanNetworking()
   buatKoneksiMenuju Receiver()
   if(kirim(START<EOF>))
   {
```

```
    jobs = isikanJobs()
    jobs += kirim(GETJOBS<EOF>)
    kirim(NUMJOBS<EOF>)
    sinkronisasi()
    kirim(WATCH<EOF>)
   }
   if (terjadiError)
   {
    kirim(ENDPROG<EOF>)
    matikanKoneksi()
   }
  }
  tungguSampaiNextSync()
 }
}
```

Pada algoritma ini aplikasi akan menginisiasi jobs, atau pekerjaan sinkronisasi per *file* dengan array kosong.Langkah selanjutnya yaitu menyalakan networking dan membuat koneksi pada *receiver* sesuai dengan IP dan port yang telah ditentukan.

Setelah itu, aplikasi akan mengirimkan *command* start kepada *receiver* untuk bersiap-siap menerima kiriman *file* dan folder dari *sender*. Aplikasi lalu mengkalkulasi *file* dan folder yang akan dikirimkan dan memasukkannya dalam array jobs yang tadi telah diinisiasi. Kemudian aplikasi akan mengirimkan *command* get jobs untuk mendapatkan sisa jobs yang belum selesai terlaksana dari *receiver*. Aplikasi mengirimkan jumlah jobs kepada *receiver* sebagai dasar dari progress bar yang akan ditampilkan ke pengguna.

Aplikasi melakukan proses sinkronisasi berdasarkan jobs yang telah diisi sebelumnya. Jika proses sinkronisasi selesai, aplikasi akan mengirimkan *command* watch pada *receiver* untuk memonitor penambahan / update / penghapusan *file* atau folder yang terjadi di *receiver*. Jika terjadi error, aplikasi akan mengirimkan *command* endprog untuk menghentikan proses sinkronisasi pada *receiver* lalu mematikan koneksi. Aplikasi akan menunggu sampai proses sinkronisasi selanjutnya dijalankan.

B. *Receiving*

Receiving merupakan proses penerimaan *file* dan folder dari *sender*. Pada langkah ini, aplikasi akan menerjemahkan *command key* yang dikirim oleh *sender* kemudian mengeksekusi setiap permintaan dari *sender*. Berikut adalah algoritmanya.

```
private void Sending
{
 inisiasiJobs()
 while (tidakAdaInterrupt)
 {
  if (menggunakanPropietaryProtocol)
  {
   nyalakanNetworking()
   buatKoneksiMenuju Receiver()
   if(kirim(START<EOF>))
   {
    jobs = isikanJobs()
    jobs += kirim(GETJOBS<EOF>)
    kirim(NUMJOBS<EOF>)
    sinkronisasi()
    kirim(WATCH<EOF>)
   }
   if (koneksiTerputus)
   {
    kirim(ENDPROG<EOF>)
    matikanKoneksi()
```

```
    }
   }
   tungguSampaiNextSync()
  }
 }
```

Pada algoritma ini, aplikasi akan menyalakan fitur listener dan membuka firewall sesuai dengan port yang ditentukan, kemudian memproses setiap *command* yang dikirimkan oleh *sender*. Untuk jenis-jenis *command* sudah dijelaskan pada bagian protokol mandiri. Jika terjadi error, aplikasi akan menutup port dan mematikan listener.

C. *Monitoring*

Monitoring merupakan proses menyalakan web server yang terhubung dengan kondisi aplikasi. Setiap request yang masuk akan diterjemahkan menjadi status aplikasi yang kemudian dikirimkan kembali kepada browser dengan format JSON.

```
private void Monitoring
{
 inisiasiJobs()
```

```
while (tidakAdaInterrupt)
{
 if (menggunakanPropietaryProtocol)
 {
  nyalakanNetworking()
  buatKoneksiMenuju Receiver()
  if(kirim(START<EOF>))
  {
   jobs = isikanJobs()
   jobs += kirim(GETJOBS<EOF>)
   kirim(NUMJOBS<EOF>)
   sinkronisasi()
   kirim(WATCH<EOF>)
  }
  if (koneksiTerputus)
  {
   kirim(ENDPROG<EOF>)
   matikanKoneksi()
  }
 }
 tungguSampaiNextSync()
```

```
}
}
```

Pada langkah ini aplikasi akan menyalakan fitur *listener* web sesuai dengan port yang telah ditentukan sebelumnya. Kemudian aplikasi akan memproses request *command* dari browser seperti melihat *status*, log, dan *buffer* yang terpakai. Fitur ini akan menerjemahkan *command* yang diterima dengan kondisi aplikasi saat sedang berjalan.

D. *Admin*

Admin merupakan bagian dari fitur *monitoring*, namun secara khusus dapat menyalakan, mematikan, serta melihat kondisi dari aplikasi-aplikasi SyncFolder yang sedang berjalan. Berikut adalah algoritmanya:

```
private void Admin
{
 listSyncFolder += masukkanKonfigurasiSyncFolder()
 while(true)
 {
  if(commandYangDatang exist in listSyncFolder)
```

```
  {
    jalankan Command PadaSyncFolder()
  }
  tampilkanStatusSetiapSyncFolder(listSyncFolder)
 }
}
```

Pada algoritma ini, setiap informasi SyncFolder yang didaftarkan pada Admin akan dimasukkan ke dalam list internal SyncFolder yang digunakan. Kemudian list ini akan diulang pada looping dengan default perintah adalah menampilkan kondisi setiap SyncFolder yang sedang berjalan, seperti status running, status on/off, dan log SyncFolder.

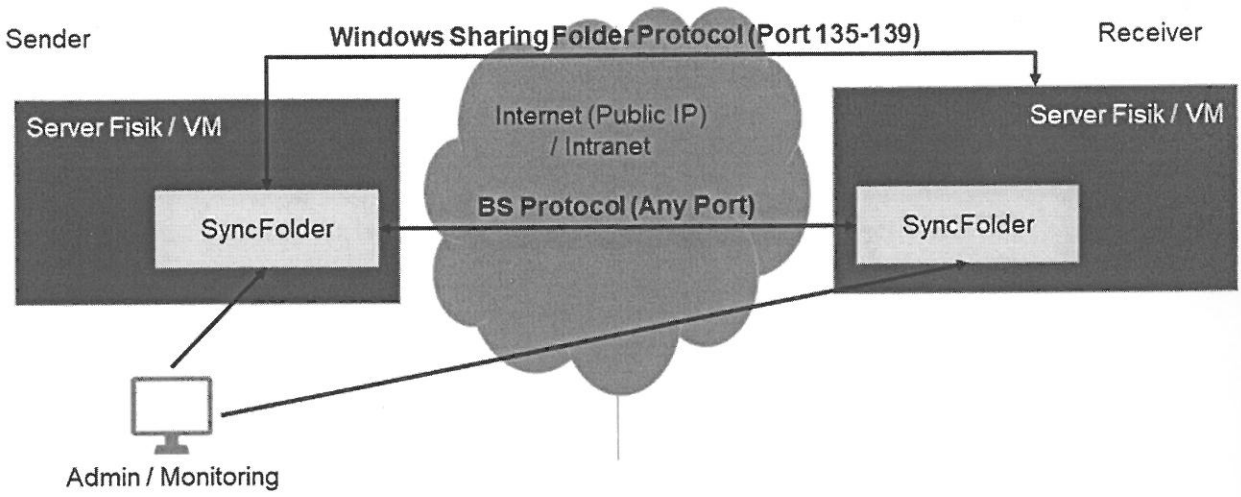
Jika terdapat *command* untuk mematikan atau menyalakan SyncFolder tertentu, algoritma ini akan melakukan pengecekan dalam list internal terlebih dahulu, lalu menjalankan *command* tersebut pada SyncFolder yang telah ditemukan.

BAB IV. DETAIL IMPLEMENTASI PROTOKOL

Dalam bab ini akan dijelaskan detail aplikasi SyncFolder sebagai implementasi dari protokol sinkronisasi, yang meliputi topologi, tampilan antarmuka dan detail program yang terdiri dari *sending and receiving, logger* dan *monitoring*, serta SyncFolderAdmin.

A. Topologi

Topologi dari aplikasi SyncFolder dapat dilihat pada gambar berikut:



Gambar 2. Topologi Aplikasi SyncFolder

Setiap aplikasi SyncFolder dapat berjalan sebagai *sender* atau sebagai *receiver*. Komunikasi antara keduanya dapat berjalan via

intranet maupun internet, jika akan dijalankan melewati internet, perlu dipastikan bahwa setiap *sender* atau *receiver* mendapatkan IP public. SyncFolder berjalan di dalam Sistem Operasi (Operating System / OS) yang berjalan pada Server maupun Virtual Machine, sehingga komunikasi antar keduanya bergantung pada settingan OS tersebut. Sebagai contoh yaitu setting firewall dan permission untuk membaca *file* atau folder. Proses sinkronisasi dengan Aplikasi SyncFolder dapat menggunakan dua mode, yaitu Windows Sharing Folder yang menggunakan port 135-139, dan mode BS Protocol (protokol mandiri) yang dapat berjalan pada port berapa saja.

Untuk dapat melakukan pengecekan aplikasi SyncFolder tanpa harus masuk ke dalam Operating System, aplikasi ini menyediakan fitur monitoring yang dapat diakses via web, ataupun aplikasi SyncFolderAdmin sebagai agregator dari aplikasi-aplikasi SyncFolder yang berjalan.

B. File dan Alur Aplikasi

Pada dasarnya, terdapat beberapa *file* yang dibutuhkan agar aplikasi SyncFolder dapat berjalan dengan baik, namun untuk mempermudah *user experience*, dibuatkan sebuah program instalasi sehingga pengguna tidak perlu mengetahui detail dari *file-file* tersebut. Selain itu, aplikasi SyncFolder juga memuat mode pengiriman

(*sending*) dan penerimaan (*receiving*) dalam satu tampilan dengan tujuan yang sama, yaitu mempermudah pengguna menjalankan aplikasi.

1. *File-file* yang diinstall

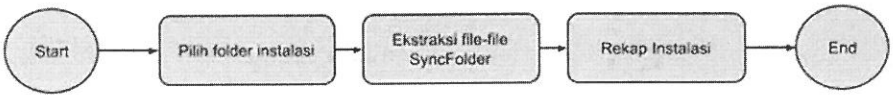
Berikut adalah beberapa *file* yang digunakan oleh SyncFolder berikut penjelasannya beserta lokasinya dalam folder.

Folder	Nama File	Keterangan
/	config.bst	Berisi konfigurasi aplikasi
	Microsoft.WindowsAPICodePack.dll	Library untuk akses *windows subsystem*
	Microsoft.WindowsAPICodePack.ExtendedLinguisticServices.dll	Library untuk akses *windows subsystem*
	Microsoft.WindowsAPICodePack.Sensors.dll	Library untuk akses *windows subsystem*
	Microsoft.WindowsAPICodePack.Shell.dll	Library untuk akses *windows subsystem*

	Microsoft.WindowsAPICodePack.ShellExtensions.dll	Library untuk akses *windows subsystem*
	Newtonsoft.Json.dll	Library untuk pemodelan data menggunakan format JSON
	SyncFolder.exe	Aplikasi utama
	SyncFolderService.exe	Aplikasi tanpa antarmuka yang berjalan dalam *windows service*
	System.Net.Http.dll	Library untuk komunikasi data
/monitoring	liveclock.js	LIbrary untuk menampilkan waktu
	syncfolder.css	Styling tampilan
	syncfolder.html	Tampilan web untuk monitoring
	syncfolder.js	Library untuk mengelolaan tampilan web
	template.html	Template web untuk monitoring

2. Alur Instalasi

Berikut adalah alur instalasi aplikasi SyncFolder:

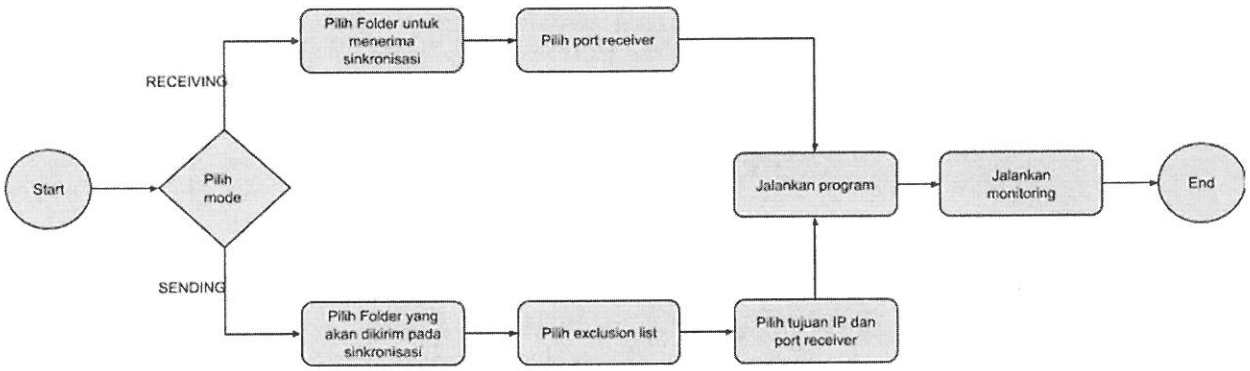


Gambar 3. Alur Instalasi SyncFolder

Instalasi dimulai dengan menjalankan program *installer*, kemudian akan disediakan pilihan untuk menginstall di folder mana SyncFolder akan diinstal. Langkah selanjutnya yaitu *installer* akan mengekstrak *file-file* yang dibutuhkan ke dalam folder yang telah dipilah. Langkah terakhir adalah menampilkan rekap instalasi.

3. Alur Aplikasi

Berikut adalah alur jalannya aplikasi SyncFolder:



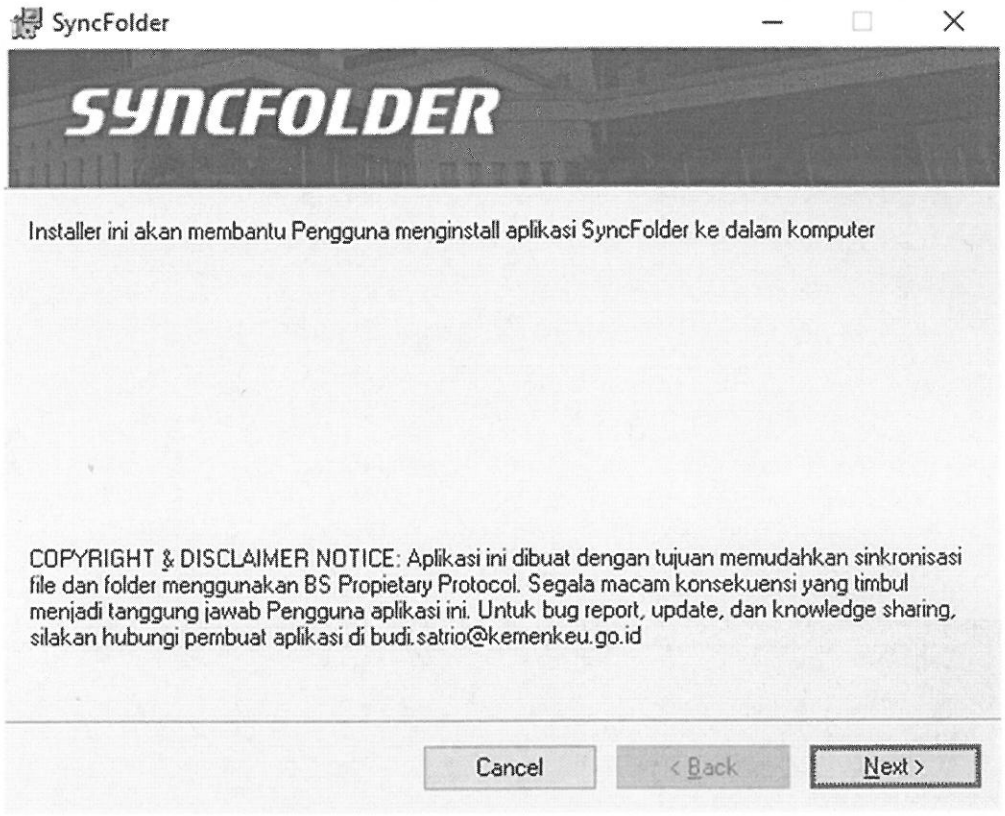
Gambar 4. Alur Aplikasi SyncFolder

Aplikasi dimulai pertama kali dengan memilih mode mana yang akan digunakan, apakah *sending* atau *receiving*. Jika yang dipilih adalah *receiving*, maka yang dilakukan adalah menentukan folder mana yang akan menerima pengiriman dari *sender*. Folder ini akan disinkronisasikan dengan folder yang berada di *sender*. Kemudian langkah selanjutnya yaitu menentukan port mana yang akan digunakan, setelah itu jalankan aplikasi dengan mengklik tombol start *receiving*. Langkah terakhir yaitu menyalakan fitur monitoring agar dapat diakses kondisi jalannya aplikasi.

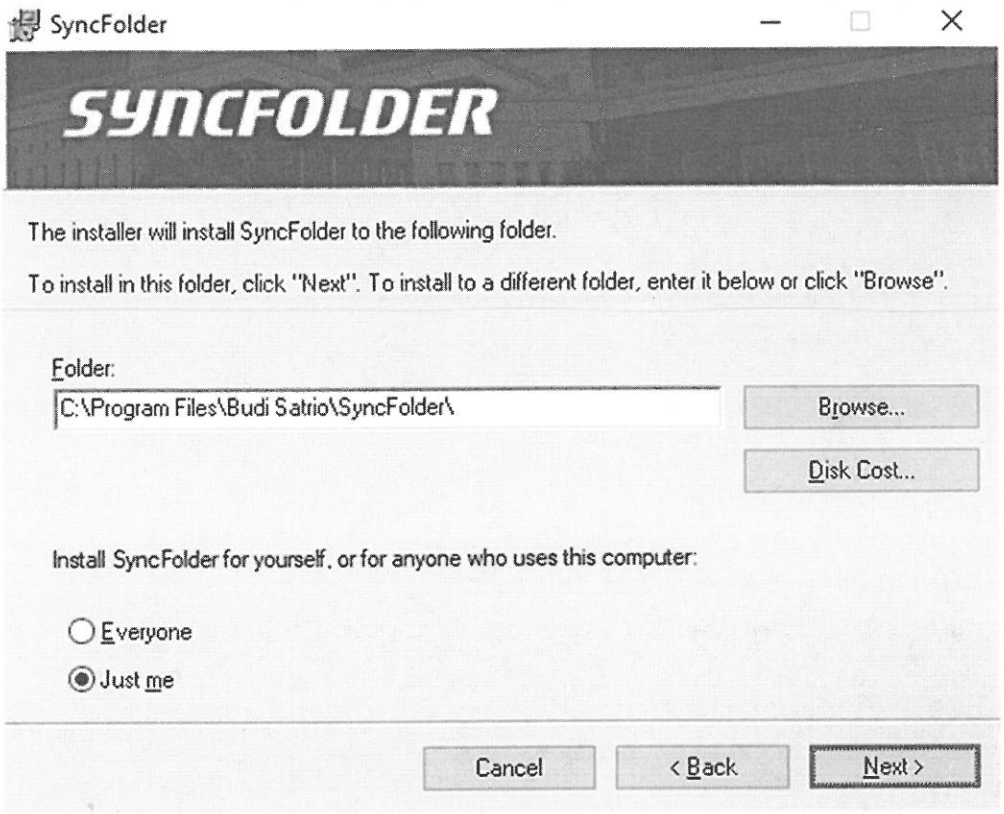
Jika yang dipilih adalah *sending*, maka yang dilakukan adalah memilih folder mana yang akan dikirim untuk disinkronisasikan dengan *receiver*. Kemudian memilih exclusion list, atau *file* dan folder mana yang tidak ikut disinkronisasikan. Langkah selanjutnya yaitu memilih tujuan IP dan port *receiver* yang dituju lalu menjalankan aplikasi dengan mengklik tombol start *sending*. Langkah terakhir yaitu menjalankan monitoring agar dapat dicek kondisi sinkronisasinya.

C. Antarmuka

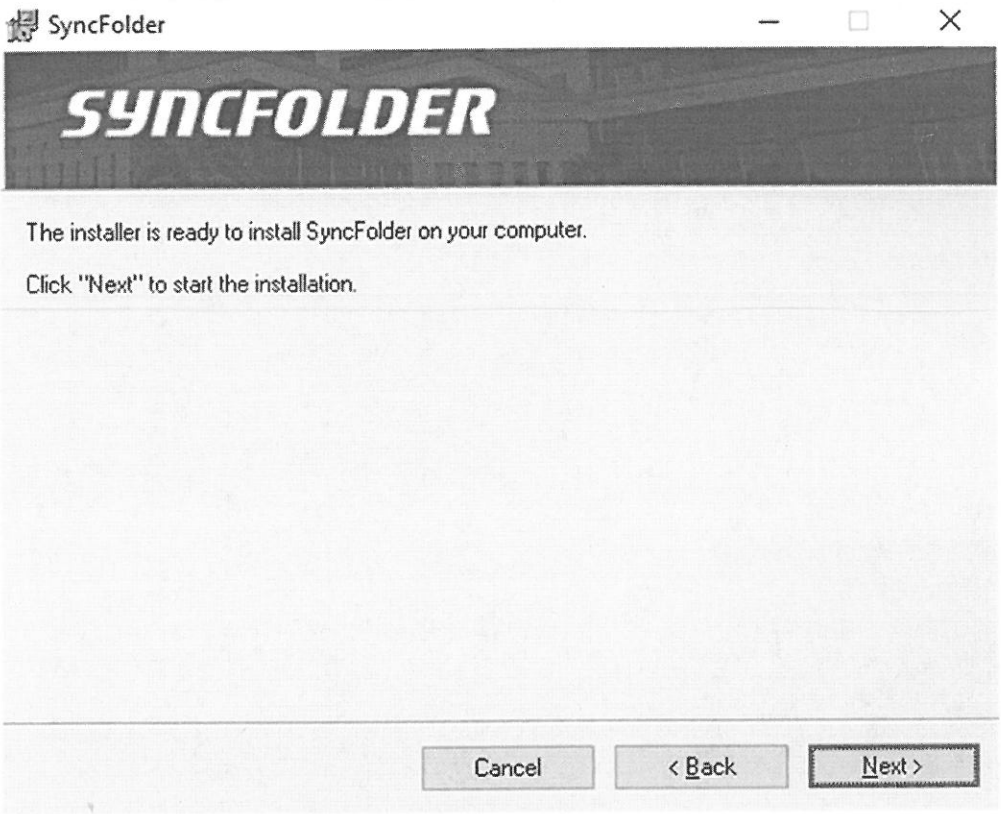
SyncFolder menggunakan installer yang umum digunakan dengan tujuan mempermudah proses instalasi aplikasi, berikut adalah tampilannya.



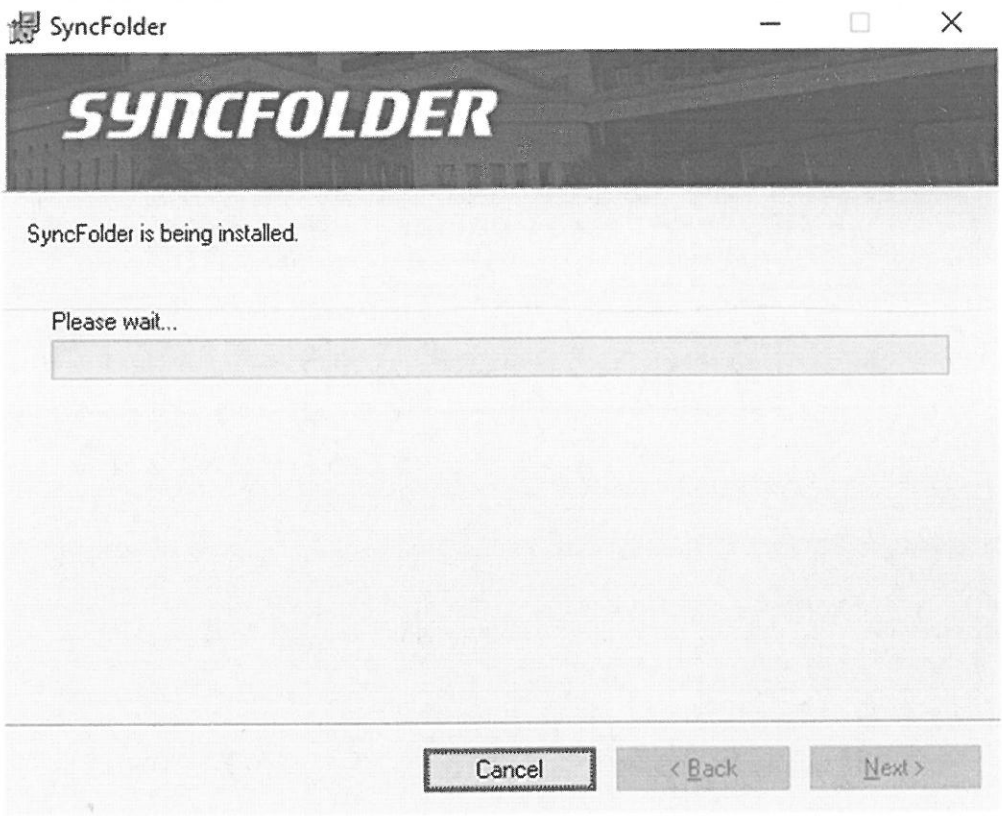
Gambar 5. Tampilan awal instalasi SyncFolder



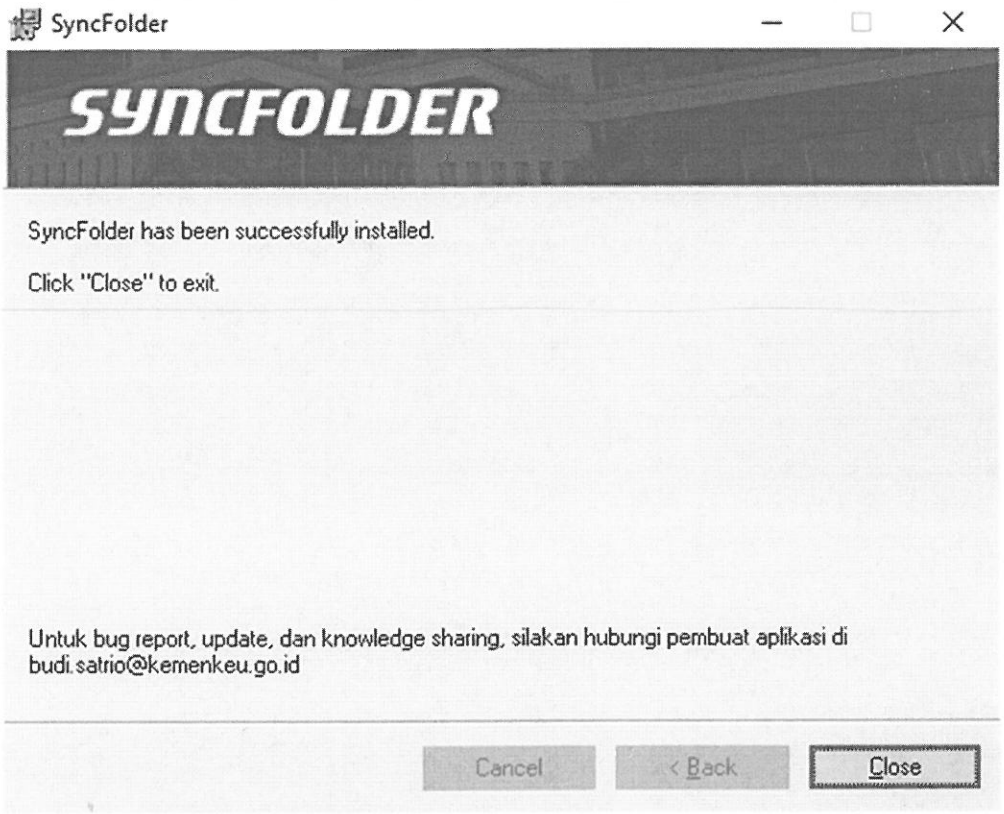
Gambar 6. Tampilan pemilihan folder untuk aplikasi SyncFolder



Gambar 7. Tampilan konfirmasi melanjutkan instalasi SyncFolder

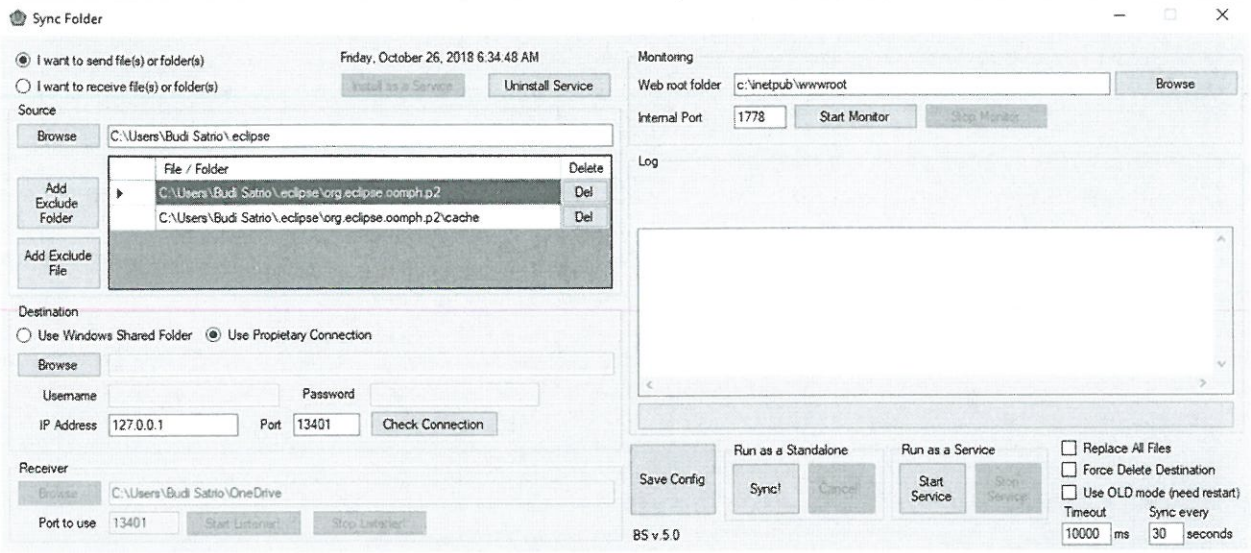


Gambar 8. Tampilan progress instalasi SyncFolder

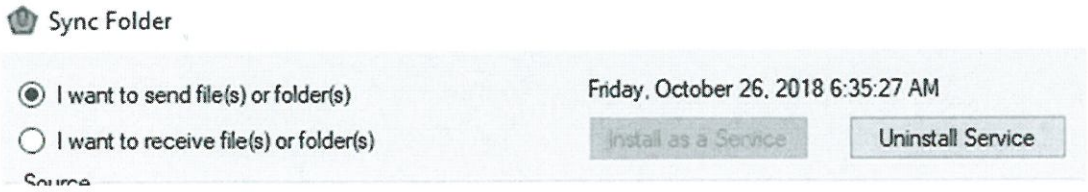


Gambar 9. Tampilan akhir proses instalasi SyncFolder

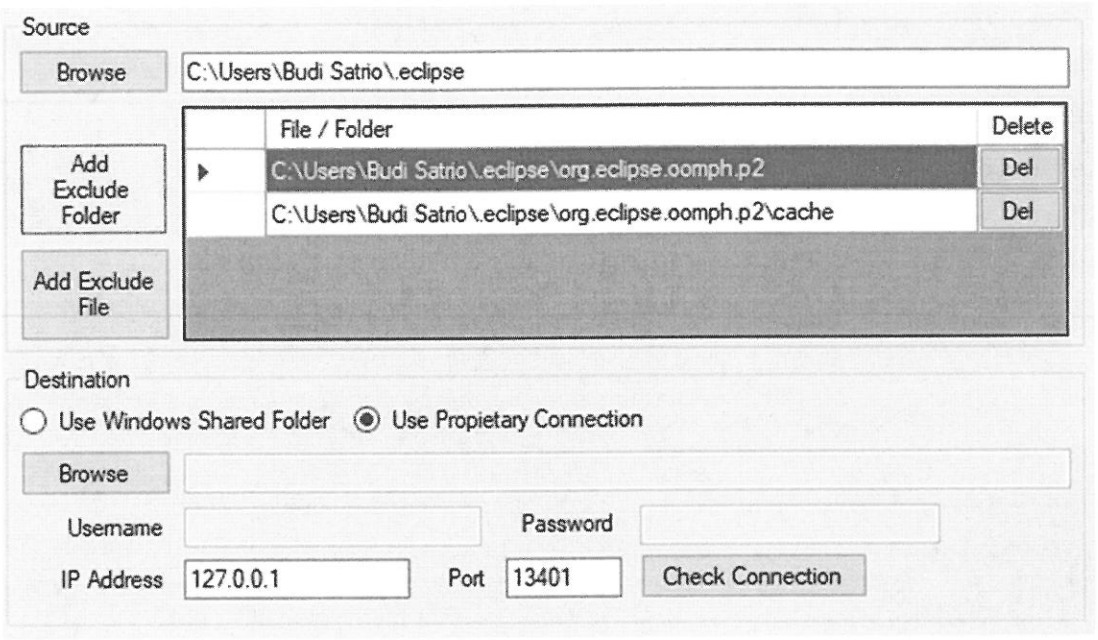
Tampilan SyncFolder memiliki konsep dasar *sending* and *receiving*, logger, dan monitoring dalam satu tampilan, sehingga tampilan utamanya adalah sebagai berikut



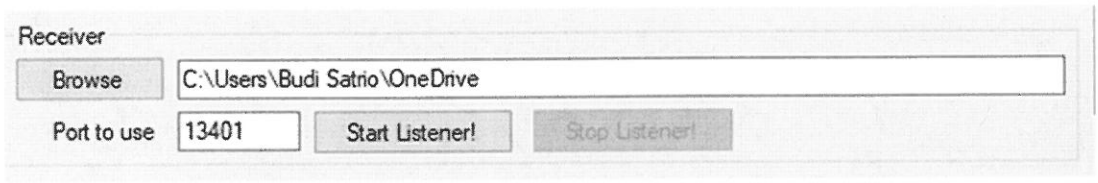
Gambar 10. Tampilan keseluruhan aplikasi SyncFolder



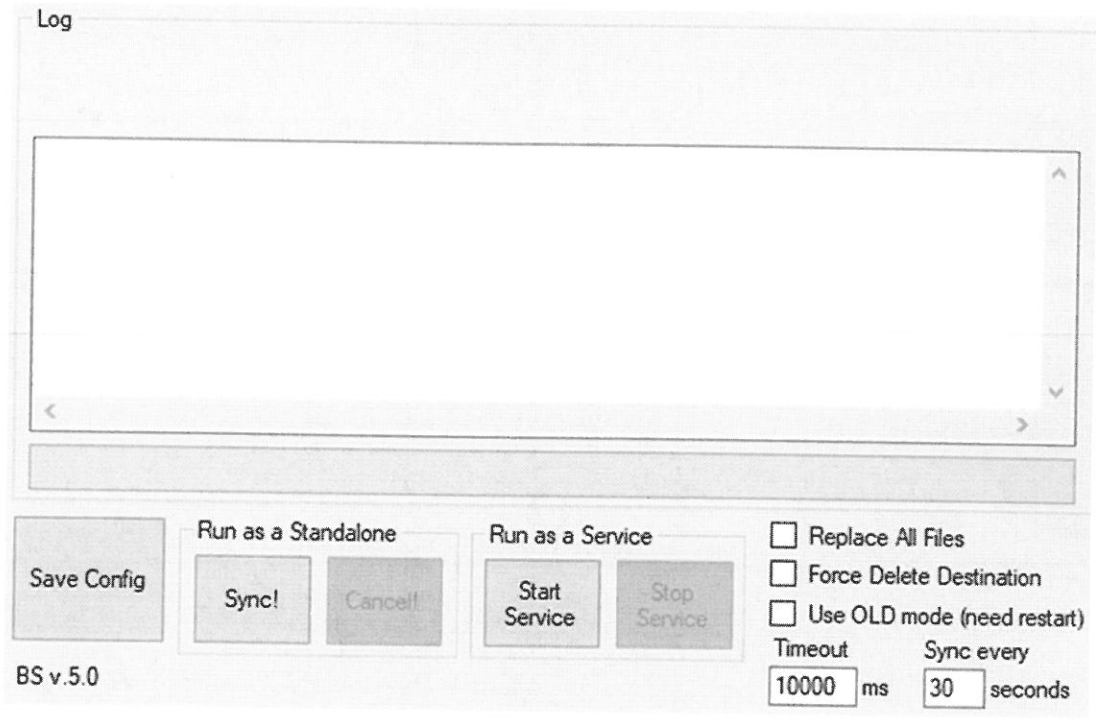
Gambar 11. Tampilan pemilihan mode aplikasi SyncFolder



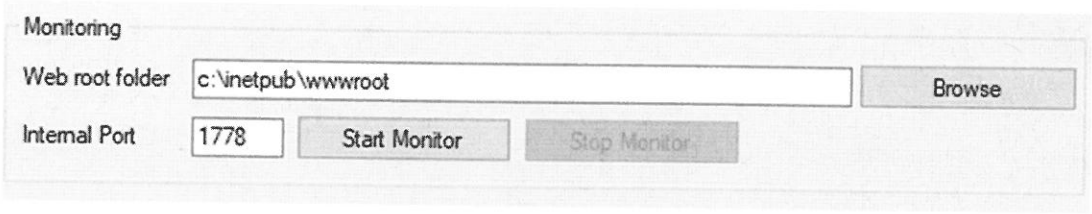
Gambar 12. Tampilan mode *sending* aplikasi SyncFolder



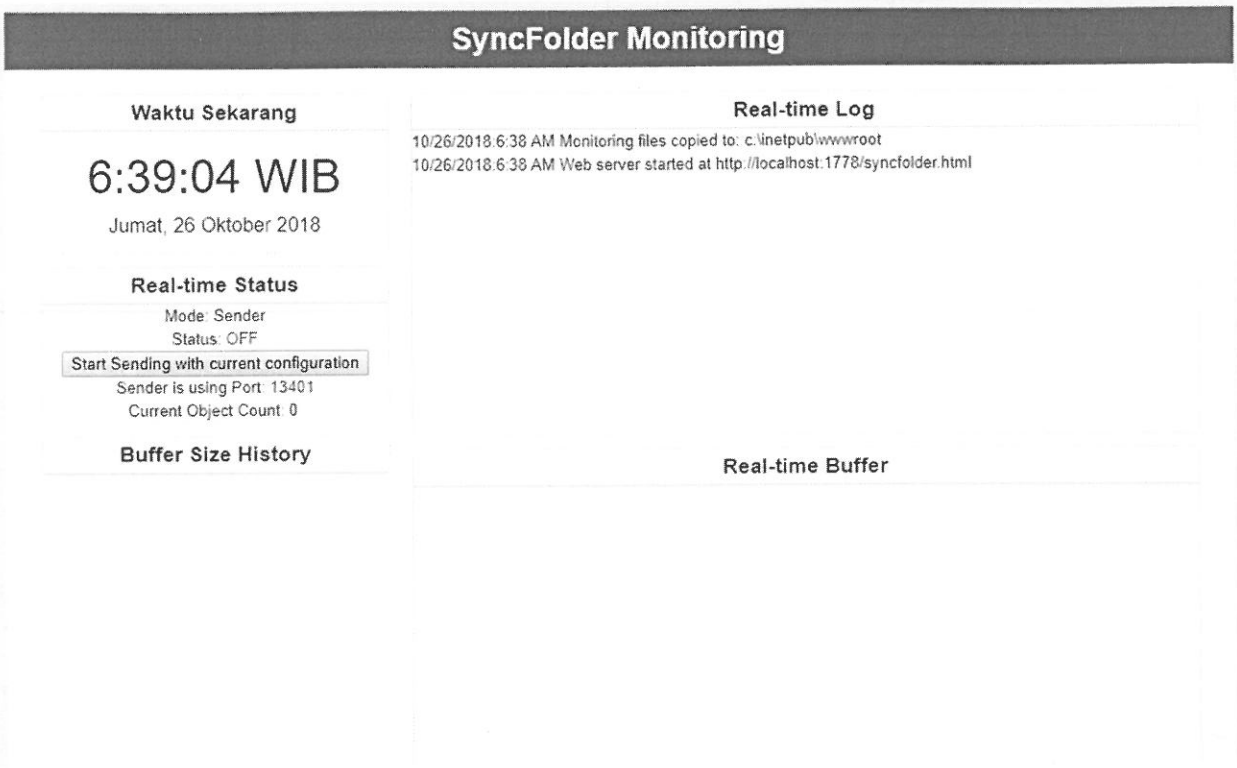
Gambar 13. Tampilan mode *receiving* aplikasi SyncFolder



Gambar 14. Tampilan log dan menjalankan aplikasi SyncFolder



Gambar 15. Tampilan monitoring aplikasi SyncFolder



Gambar 16. Tampilan web monitoring aplikasi SyncFolder

Gambar 17. Tampilan SyncFolderAdmin yang memanage aplikasi-aplikasi SyncFolder yang sedang berjalan

D. *Sending & Receiving*

Seperti yang sudah dijelaskan pada bagian algoritma, proses pengiriman dan penerimaan ini akan berjalan dalam *background* sehingga tidak mengganggu antarmuka dari aplikasi. Setiap progress pengiriman akan dilaporkan dalam bentuk progress bar di aplikasi, baik itu mode *sending* maupun *receiving*.

Pada mode *sending*, yang pertama dilakukan adalah memilih folder yang akan disinkronkan. Semua folder dan *file* dalam folder tersebut akan disinkronkan satu per satu (recursive tree), oleh karena itu jika terdapat folder atau *file* yang tidak ingin disinkronkan, dapat memilih excluded folder atau excluded *file*. Langkah selanjutnya adalah memilih protokol yang digunakan, apakah itu Windows Sharing Folder, atau menggunakan protokol mandiri. Jika menggunakan protokol mandiri, maka cukup memasukkan IP dan port dari *receiver*. Jika menggunakan Windows Sharing, maka masukkan URL tempat sharing folder tujuan. Langkah terakhir adalah menjalankan sinkronisasi dengan menekan tombol *Start Sync*.

Terdapat beberapa opsi pada mode *sending*, diantaranya yaitu Replace All Files yang akan menimpa semua *file* di *receiver* tanpa memperhatikan modified date, Force Delete Destination yang akan menghapus *file* atau folder yang tidak sesuai dengan yang dikirim dari

sender, Use Old Mode yang akan menjalankan aplikasi tanpa menggunakan Delta (Proses pengiriman selisih *file* atau folder dari yang sudah berada pada *receiver*), Timeout yang akan membatalkan pengiriman apabila *receiver* tidak memberikan jawaban, serta *Sync Every* yang akan otomatis menjalankan lagi sinkronisasi sesuai dengan inputan waktu dalam detik yang dimasukkan.

Pada mode *receiving*, langkah yang dilakukan lebih mudah yaitu dengan memilih folder yang akan menerima *file* dan folder dari *sender*, lalu memilih port yang akan dibuka pada *receiver*. Langkah terakhir yaitu menekan tombol *start listener*.

E. Logger

Logger ini berfungsi untuk mencatat segala aktifitas yang berjalan pada SyncFolder, mulai dari memulai aplikasi, memulai membuka koneksi, mengirimkan *file* dan folder, informasi monitoring, serta informasi exception handling jika terdapat error pada aplikasi.

Format dari *logger* ini adalah tanggal dan waktu, kemudian pesan, sebagai contoh adalah informasi log monitoring berikut:

```
10/26/2018:7:57 AM Monitoring files copied to: c:\inetpub\wwwroot
```

10/26/2018:7:57 AM Web server started at http://localhost:1778/syncfolder.html

Dengan adanya informasi tanggal dan waktu, pengguna aplikasi dapat menelusuri kejadian saat log tersebut ditulis untuk keperluan analisis.

F. *Monitoring*

Monitoring adalah fitur aplikasi SyncFolder untuk melihat informasi buffer, log, serta status jalannya aplikasi melalui tampilan web. Pada fitur ini, pengguna dapat memasukkan sendiri port yang akan digunakan, sehingga tidak terikat bahwa web harus berjalan pada port 80 atau 443 untuk alasan keamanan. Untuk menggunakan fitur ini, pengguna dapat memasukkan url folder dari web server yang sudah ada, jika monitoring mau ditempelkan dengan aplikasi web yang sudah berjalan, atau cukup dengan memasukkan portnya saja, sehingga fitur monitoring dapat diakses dengan menggunakan port yang sudah ditentukan.

G. SyncFolderAdmin

SyncFolder Admin merupakan aplikasi selain SyncFolder yang sifatnya untuk melakukan administrasi aplikasi-aplikasi SyncFolder yang sedang berjalan. Sesuai dengan ruang lingkup, penulis tidak

menjelaskan secara detail mengenai aplikasi ini, namun cukup dengan garis besarnya saja. SyncFolderAdmin memiliki fitur untuk melihat log, menyalakan, mematikan, serta melihat informasi SyncFolder yang sedang berjalan. Jika aplikasi SyncFolder mati atau terjadi error, SyncFolderAdmin akan menampilkan warna merah pada aplikasi tersebut dan menampilkan fitur untuk menyalakan SyncFolder secara remote.

Aplikasi ini berguna bagi tim manajemen aplikasi untuk mengetahui apakah sinkronisasi berjalan secara optimal sekaligus melihat status dari setiap SyncFolder yang sedang digunakan.

BAB V. KEUNGGULAN DAN TANTANGAN

A. Keunggulan

Beberapa keunggulan aplikasi SyncFolder ini adalah sebagai berikut:

1. Gratis dan Support Cepat

Aplikasi SyncFolder tidak berbayar karena dibuat oleh penulis yang merupakan pegawai Kementerian Keuangan. Support juga cepat karena penulis mudah ditemui dan dapat koordinasi secara langsung tanpa terkendala oleh lokasi dan channel komunikasi.

2. Ringan dan Simpel

Aplikasi SyncFolder hanya berukuran 210 Kb dan mudah di copy atau transfer menggunakan media USB Drive, Email, maupun WhatsApp. Instalasi yang mudah serta struktur program yang tidak rumit dan tidak menggunakan registry. Sehingga apabila program akan dihapus, cukup dengan menguninstall atau menghapus root folder dari program.

3. **Port Komunikasi yang Fleksibel**

Aplikasi SyncFolder tidak harus berjalan dengan port tertentu sehingga memudahkan pengelola aplikasi dan jaringan ketika akan melakukan sinkronisasi. Baik dalam area Demilitarized Zone, Intranet dalam Data Center, maupun komunikasi antara DC dan DRC, SyncFolder mampu menembus jaringan tersebut selama port yang digunakan oleh protokol mandiri tersebut tidak masuk ke dalam area hardening atau firewall jaringan.

4. ***Monitoring* dan *Admin***

Aplikasi SyncFolder dapat dimonitor secara remote dengan menggunakan interface web, sehingga pengguna aplikasi tidak perlu masuk ke dalam server hanya untuk mengecek kondisi jalannya SyncFolder. Monitoring dapat dilakukan melalui web desktop, ataupun dari mobile dengan menggunakan browser.

Untuk melakukan administrasi, disediakan pula SyncFolderAdmin sehingga pengelola aplikasi tidak perlu masuk ke dalam server untuk mematikan atau menyalakan SyncFolder, maupun untuk mengecek kondisi aplikasi termasuk log nya.

5. ***Exclusion* Files & Folder**

Aplikasi SyncFolder memiliki fitur untuk mengecualikan *file* dan folder yang telah ditentukan sebelumnya. Hal ini sangat berguna bagi pengelola aplikasi karena *file-file* setting seperti web.config ataupun folder TEMP tidak harus disinkronisasi, karena ukuran folder bisa menjadi sangat besar, serta TEMP digenerate oleh masing-masing server secara unik. Jika folder tersebut ikut disinkronisasikan, maka dapat mengganggu jalannya aplikasi web pada server tersebut.

6. **Performa Cepat dan Lebih Kaya Fitur**

Dari testimoni pengguna aplikasi SyncFolder, dapat diketahui bahwa aplikasi ini berkinerja cepat dan memiliki fitur-fitur yang dibutuhkan oleh pengelola aplikasi. Salah satu testimoni adalah "Aplikasi SyncFolder ini lebih cepat daripada buatan Mi**o**ft, buktinya saat sinkronisasi portal dari DC menuju DRC, SyncFolder berjalan dengan sangat cepat, seolah-olah tidak ada firewall".

Dari analisis teknis, performa cepat bisa terjadi karena protokol mandiri yang digunakan sangat simpel dan bisa menggunakan port yang bebas sehingga tidak terpengaruh oleh Application Filter dari sisi Jaringan.

7. Protokol Mandiri

Keunggulan utama dari aplikasi ini adalah penggunaan protokol mandiri yang lebih secure dan efektif karena dibuat oleh orang internal Kementerian Keuangan sehingga terjamin keamanannya dan siap untuk diupdate jika dibutuhkan penambahan fitur.

B. Tantangan

Beberapa tantangan yang dihadapi aplikasi ini adalah sebagai berikut:

1. Pembuatan Hak Cipta

Protokol mandiri ini akan dapat ditiru dan bahkan digunakan oleh perusahaan komersial untuk menjual produknya apabila tidak dilindungi oleh Hak Cipta. Oleh karena itu, tantangan yang harus dijalankan ke depan adalah membuatkan Hak Cipta dari protokol mandiri ini.

2. Bug Solving

Karena aplikasi SyncFolder ini dibuat oleh perorangan, akan sangat mungkin muncul bug yang terjadi saat digunakan di area production. Oleh karena itu, dibutuhkan partisipasi aktif antara pengguna aplikasi dan penulis untuk selalu melaporkan bug serta informasi environment saat bug terjadi. Sampai saat

ini, penulis telah memetakan dan memperbaiki puluhan Bug yang dilaporkan oleh pengguna aplikasi.

3. Pengembangan Protokol

Tantangan selanjutnya dari protokol mandiri ini adalah memperbaiki dan bahkan mengupdate protokol untuk menjadi lebih cepat, lebih stabil, dan dapat digunakan oleh aplikasi lainnya. Sehingga, ke depannya protokol mandiri ini dapat digunakan oleh aplikasi web lain, tanpa harus melalui SyncFolder untuk mengirim dan menerima *file* atau folder.

BAB VI. KESIMPULAN, SARAN, DAN PENUTUP

A. Kesimpulan

Aplikasi SyncFolder ini dibangun untuk membantu proses bisnis pengelolaan aplikasi baik di Data Center maupun Disaster Recovery Center Kementerian Keuangan. Di dalamnya terdapat penggunaan protokol mandiri yang dibuat dengan tujuan melakukan sinkronisasi secara lebih efektif dan efisien. Protokol mandiri tersebut menggunakan metode *half duplex* dengan *command key* yang terkompress untuk mempercepat komunikasi. Algoritma sinkronisasi yang digunakan aplikasi SyncFolder meliputi *sending*, *receiving*, monitoring, dan admin yang di support dengan tampilan antarmuka dan instalasi yang mudah.

Terdapat banyak keunggulan aplikasi ini serta tantangannya ke depan. Keunggulan dan tantangan utama adalah mengenai protokol mandiri. Keunggulannya yaitu protokol ini simpel dan secure, namun tantangan kedepannya adalah perlunya perlindungan hak cipta agar tidak ditiru oleh perusahaan komersial.

B. Saran

Perlunya tim khusus yang menangani implementasi protokol sinkronisasi ini sehingga setiap masalah akan lebih mudah diketahui secara lengkap dan diperbaiki dengan cepat. Selain itu, aplikasi ini harus mendapatkan Hak Cipta secepatnya agar tidak mudah dikopi oleh pihak ketiga.

C. Penutup

Dengan semakin digunakannya aplikasi ini di Data Center Kementerian Keuangan, penulis membutuhkan masukan-masukan baik dari segi teknis, keamanan, maupun user experience. Diharapkan dengan banyaknya masukan tersebut, aplikasi ini dapat mempermudah proses bisnis Pusintek dengan sistem keamanan yang secure.

DAFTAR PUSTAKA

[1] Howells, George A., James A. Murray, and Douglas E. Woodman. "Digital duplex transmission system." U.S. Patent 4,220,816, issued September 2, 1980.

[2] Durumeric, Zakir, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver et al. "The matter of heartbleed." In *Proceedings of the 2014 conference on internet measurement conference*, pp. 475-488. ACM, 2014.