IMPLEMENTASI REPLIKASI FILE SERVER DENGAN METODE *CHUNKING* MENGGUNAKAN *UNSTRUCTURED DATABASE*

Disusun Oleh:

N a m a : Elbama Giofandra R. R

NIM : 16523233

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2020

HALAMAN PENGESAHAN DOSEN PEMBIMBING

IMPLEMENTASI REPLIKASI FILE SERVER DENGAN METODE *CHUNKING* MENGGUNAKAN *UNSTRUCTURED DATABASE*

TUGAS AKHIR

Disusun Oleh:

N a m a : Elbama Giofandra R. R

NIM : 16523233

Yogyakarta, 24 Juli 2020

Pembimbing,

(Mukhammad Andri Setiawan, S.T, M.Sc, Ph.D.)

HALAMAN PENGESAHAN DOSEN PENGUJI

IMPLEMENTASI REPLIKASI FILE SERVER DENGAN METODE *CHUNKING* MENGGUNAKAN *UNSTRUCTURED DATABASE*

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 24 Juli 2020

Tim Penguji

Dr. Mukhammad A Setiawan, S.T., M.Sc. _____

Anggota 1

Dr. Raden Teduh Dirgahayu, S.T., M.Sc. _____

Anggota 2

Septia Rani, S.T., M.Cs. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Elbama Giofandra R. R

NIM : 16523233

Tugas akhir dengan judul:

IMPLEMENTASI REPLIKASI FILE SERVER DENGAN METODE *CHUNKING* MENGGUNAKAN *UNSTRUCTURED DATABASE*

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 24 Juli 2020

(Elbama Giofandra)

Dipindai dengan CamScanner

HALAMAN PERSEMBAHAN

Alhamdulillah hirobbil'alamin, segala puji bagi Allah SWT yang telah melimpahkan ridho, rahmat, dan hidayahnya. Sholawat serta salam tetap tercurahkan kepada Nabi Muhammad SAW, yang telah menjadi suri tauladan bagi seluruh umat manusia di muka bumi ini. Semoga tugas akhir ini menjadi bentuk ikhtiar untuk mendapat ilmu yang diridhoi Allah SWT. Dengan mengucapkan hamdallah, tugas akhir ini dipersembahkan kepada:

1. Kedua orangtua tercinta, Bapak Sugiyo dan Ibu Sriwatiningsih yang dengan tulus hati membesarkan dan mendidik anaknya hingga saat ini dan selalu memberikan doa, dukungan, dan motivasi.
2. Kakak tersayang saya Elgiva Giowanda Dhussa, yang selalu memberikan dukungan dalam bentuk apapun hingga saat ini.
3. Bapak Mukhammad Andri Setiawan S.T., M.Sc., Ph.D selaku dosen pembimbing yang telah memberikan arahan dan bimbingan terbaik, sehingga tugas akhir ini dapat selesai.
4. Jurusan Informatika yang telah menjadi wadah bagi mahasiswa untuk menuntut ilmu dengan benar.
5. Semua jajaran dosen Informatika yang telah memberikan ilmu kepada semua mahasiswanya.
6. Teman-teman saya yang selalu menemani ketika susah maupun senang. Mereka menjadi teman terbaik dalam menggapai mimpi besar saya.
7. Seluruh pihak yang tidak bisa disebutkan satu per satu, terimakasih atas semuanya.

HALAMAN MOTO

“Allah tidak akan membebani seseorang melainkan dengan kesanggupannya”
(Qs Al Baqarah 286)

“Maka Maha Tinggi Allah Raja Yang sebenar-benarnya, dan janganlah kamu tergesa-gesa membaca Al qur’an sebelum disempurnakan mewahyukannya kepadamu, dan katakanlah: “Ya Tuhanku, tambahkanlah kepadaku ilmu pengetahuan”
(Qs At Thaha 114)

“Jika kau menungguku untuk menyerah, kau akan menungguku selamanya”
(Uzumaki Naruto)

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakatuh

Alhamdulillah, segala puji dan syukur saya panjatkan kepada Allah SWT, yang telah memberikan ridho dan rahmatnya. Sehingga penulis dapat menyelesaikan tugas akhir berjudul "Implementasi Replikasi File Server Dengan Metode *Chunking* Menggunakan *Unstructured Database*" yang merupakan bentuk ikhtiar saya dalam menuntut ilmu. Laporan tugas akhir ini disusun untuk memenuhi syarat kelulusan pendidikan Strata 1 (S1) Jurusan Informatika Universitas Islam Indonesia. Penulis menyadari bahwa laporan tugas akhir ini tidak akan mencapai tahap akhir tanpa dukungan dari beberapa pihak. Oleh karena itu, penulis akan menghaturkan rasa terima kasih sebesar-besarnya kepada:

1. Allah SWT, atas segala limpahan rahmat dan karunia-Nya yang selalu menemani langkah manusia dalam segala hal di bumi ini. Termasuk dalam hal ridho untuk menyelesaikan tugas akhir saya.
2. Kedua orangtua saya, Bapak Sugiyo dan Ibu Sriwatiningsih atas ketulusan hati untuk membimbing dan memberikan dukungan hingga saat ini.
3. Bapak Fathul Wahid, S.T., M.Sc., Ph.D., selaku Rektor Universitas Islam Indonesia
4. Bapak Prof. Dr. Ir. Hari Purnomo, M.T., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Bapak Mukhammad Andri Setiawan S.T., M.Sc., Ph.D selaku dosen pembimbing.
6. Bapak dan Ibu dosen Program Studi Informatika, yang telah memberikan ilmu yang bermanfaat kepada saya. Semoga Allah SWT selalu melindungi mereka dalam membimbiing mahasiswanya.
7. Teman-teman saya terutama grup "RM. Andalas" yang selalu menemani dan menghibur saya selama masa perkuliahan.
8. Kakak saya tercinta Elgiva Giowanda Dhussa, yang selalu mengalah dari saya kecil hingga dewasa. Berkatnya dukungan dan kasih sayangnya saya dapat menjadi pribadi yang lebih baik.
9. Teman-teman Hexadecima angkatan 2016, dan kakak angkatan yang tidak bisa disebutkan satu per satu.
10. Teman-teman SMA saya, yang selalu menjadi sahabat sejak saya menempuh sekolah menengah atas hingga saat ini. Terimakasih telah menjadi sahabat yang ada dalam suka maupun duka.

11. Arif Nur Khoirudin, yang telah menjadi guru tersabar dan terbaik saya di luar kampus.
12. Semua pihak yang tidak dapat saya sebutkan satu per satu, terimakasih atas semua yang telah diberikan kepada saya.

Semoga segala bentuk dukungan mendapatkan imbalan dari Allah SWT baik secara langsung maupun tidak. Penulis memohon maaf sebesar-besarnya apabila selama menyelesaikan tugas akhir ini terdapat kesalahan. Penulis dengan segala kekurangan nya mengharapkan manfaat dari pembelajaran yang didapat. Semoga laporan tugas akhir ini bermanfaat bagi pihak-pihak yang membutuhkannya.

Yogyakarta, 24 Juli 2020

(Elbama Giofandra)

SARI

Meningkatnya angka akses internet di Indonesia yang telah mencapai 64,8 persen dari jumlah penduduk. Dengan bervariasinya kondisi internet dari pengakses ini, memunculkan beberapa tantangan pada sistem yang digunakan. Tantangan ini menghadirkan permintaan untuk menciptakan sistem yang dapat digunakan dalam kondisi internet yang tidak memadai. Pengguna dalam kondisi internet yang tidak memadai, sering mengalami kegagalan pengunggahan berkas. Penelitian ini mencoba untuk memunculkan beberapa solusi dalam menyelesaikan masalah yang ada. Metode *chunking* adalah skema untuk memotong file menjadi beberapa potongan kecil dan masing-masing potongan memiliki ukuran file yang sama. Metode ini mampu membuat sebuah file dengan kapasitas besar menjadi beberapa file dengan kapasitas kecil. Metode *chunking* juga memungkinkan adanya proses unggah secara berkala atau *resumable* dikarenakan potongan file dikirim satu demi satu. Selain dalam masalah kegagalan pengunggahan berkas, pengguna sering mendapati sistem dengan tingkat ketersediaan berkas yang rendah. Hal ini memiliki arti bahwa sistem konvensional memiliki media penyimpanan yang buruk. Karena tidak mampu melayani permintaan pengguna dengan cepat. Dalam menangani masalah ini diperlukan *database* yang mampu mengelola berkas dengan cepat. *Unstructured database* menjadi unggulan banyak perusahaan besar dalam menyelesaikan masalah ketersediaan file. Dengan beberapa fitur yang tidak dimiliki oleh basis data konvensional, *unstructured database* datang dengan teknologi yang lebih maju. Rancangan sistem menggunakan setiap metode, bagian, dan teknologi yang dapat diintegrasikan dalam menghadapi masalah yang ada. Rancangan sistem ini menghasilkan terobosan yang telah diuji dalam aspek-aspek sesuai dengan permasalahan yang diangkat. Sistem dengan metode chunking memiliki kemampuan untuk melayani pengguna dalam kondisi internet yang unstable dan unreliable. Rancangan sistem diketahui kemampuannya dari hasil pengujian sesuai dengan scenario permasalahan yang diangkat.

Kata kunci: *chunking; Unstructured Database; File Upload; High-availability.*

GLOSARIUM

Database	Basis data atau media penyimpanan.
Chunking	Metode pemotongan/pemecahan file.
High-availability	Tingkat ketersediaan file yang tinggi.
Unstable Internet	Internet yang tidak stabil dikarenakan beberapa alasan.
Unreliable	Internet yang tidak memadai untuk melakukan pengiriman berkas.
Http	Protokol jaringan yang termasuk dalam lapisan *application layer*.
Server	Program yang ditujukan untuk melayani permintaan.

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Berdasarkan data International Telecommunication Union (ITU), pada tahun 2019 akses terhadap internet telah mencapai angka 4 Miliar lebih di seluruh dunia (survei ITU, 2019). Berkembangnya teknologi dengan pesat dan tingginya akses internet memunculkan tantangan terhadap arsitektur sistem yang menjadi layanan bagi pengguna internet. Dimana hal tersebut dipengaruhi oleh perancangan *repository* dan *server* pada sebuah sistem yang diakses. Secara langsung perkembangan tersebut berpengaruh terhadap permintaan untuk menghadirkan arsitektur sistem yang dapat digunakan dalam keadaan pengguna yang bervariasi. Kebutuhan terhadap penyimpanan data yang besar pun meningkat apabila meninjau dari data akses di atas. Jenis-jenis file yang ditransfer oleh pengguna pun semakin besar dan bervariasi. Menurut survei yang dilakukan oleh Asosiasi Penyelenggara Jasa Internet Indonesia (APJII) pada tahun 2018, pengguna internet mencapai angka 171.176.716 atau sekitar 64,8 persen dari jumlah penduduk di Indonesia. APJII juga memaparkan bahwa sebesar 96,6 persen pengguna internet terhubung dengan internet melalui paket data operator seluler atau disebut *broadband* internet (survei APJII, 2018). Hal ini menjelaskan bahwa banyak dari pengguna internet di Indonesia bergantung kepada kondisi internet dari ISP atau *Internet Service Provider*. Sedangkan standar *Internet Service Provider* di Indonesia menggunakan konsep broadband. Konsep utama dari *broadband* internet adalah *sharing bandwidth*, dengan kata lain membagi jalur komunikasi dengan pengguna ISP lainnya. Hal ini menyebabkan kondisi internet pengguna yang menggunakan *broadband* internet menjadi fluktuatif dan tidak terjamin. Pada *broadband* internet juga sering terjadi ketidakseimbangan antara kecepatan unggah dan unduh (Srikanth, et al.,2011). Faktor inilah yang menyebabkan diperlukannya perancangan sistem yang dapat melayani kondisi setiap pengguna internet yang menggunakan koneksi tidak stabil.

Ada beberapa faktor lain yang memperkuat adanya tantangan tersebut yaitu bervariasinya kondisi pengakses internet. Banyak ditemukan kondisi konektivitas yang *unreliable* dan *unstable* yang menyebabkan lemah atau hilangnya konektivitas pada saat proses unggah dan unduh data. Salah satu faktornya adalah keadaan seperti lokasi pengakses internet yang berada pada kawasan yang jauh dari jangkauan *access-point* ataupun BTS (*Based Transceiver Station*) (Ilker, et al.,2017). Maupun bebrapa factor seperti cuaca yang buruk, jumlah pengguna internet

yang banyak, kecilnya bandwidth yang disediakan oleh provider, dan masih banyak lagi kondisi-kondisi yang menyebabkan pengguna memiliki internet yang buruk. Namun pada faktanya dalam keadaan internet yang stabil dan memadai, masih sering terjadi koneksi yang terputus secara mendadak maupun terjadinya *error*. Hal ini sangat berpengaruh apabila pengguna sedang melakukan proses *upload* file pada sebuah *repository* atau *database*. Faktor-faktor di atas memperkuat tantangan yang semakin penting untuk diselesaikan. Menciptakan arsitektur sistem *repository* maupun rancangan *server* yang fleksibel terhadap masalah yang ada menjadi sangat penting untuk dilakukan.

Ditinjau dari angka akses pengguna terhadap internet yang semakin meningkat memunculkan kebutuhan baru untuk menyediakan penyimpanan data yang besar. Hal ini juga diperkuat dengan semakin besarnya ukuran file yang diunggah oleh pengguna. Juga semakin bervariasinya kondisi pengguna ketika mentransfer file yang tidak jarang menyebabkan file gagal diunggah. Oleh karena itu dibutuhkan suatu sistem yang *reliable* terhadap tantangan-tantangan yang ada. Tantangan yang dihadapi adalah:

a. Terputusnya konektivitas saat proses unggah file dilakukan. Hal ini bisa dikarenakan besarnya kapasitas file yang diunggah terlalu besar sehingga membutuhkan kecepatan internet yang besar juga. Beberapa sistem melakukan pendekatan dengan membatasi ukuran file sehingga pengguna dipaksa untuk mengunggah file yang berukuran kecil (*file-compress*).
b. *Http-timeout* yang sering terjadi ketika pengguna sedang mengunggah file berkapasitas besar, Hal ini dikarenakan file besar yang diunggah membutuhkan koneksi terhadap *http* yang cukup lama. Sehingga potensi untuk bermasalah sangat besar.
c. *High availability* dan *load-balancing* terhadap layanan unduh file. Dalam arsitektur sistem yang konvensional, sering terjadi *overload* dikarenakan hanya memiliki satu jalur untuk menangani trafik (Dildar, et al, ,2019).
d. *Resumable-process* ketika terjadi masalah ketika proses unggah berlangsung. Hal ini memuat pengguna tidak perlu untuk mengunggah file dari awal ketika koneksi terputus saat proses unggah terjadi.

Beberapa penelitian sebelumnya telah berusaha memunculkan kajian tentang masalah serupa. Terutama tentang metode yang digunakan untuk mengunggah file agar lebih maksimal.

Beberapa kajian telah menganalisa bahwa banyak terjadi kelemahan dengan sistem yang mengunggah file secara langsung. Dikarenakan ketika kondisi internet yang *unstable* dan *unreliable* akan besar kemungkinan terjadi kegagalan pemindahan file ketika *lost-connection*. Sehingga apabila proses pemindahan berhenti maka pengunggahan file akan gagal. Hal ini membuktikan perlu adanya penanganan terhadap file pada proses pemindahan ke *repository* dalam kondisi internet yang tidak stabil. Dalam kondisi seperti ini penangananan terhadap file harus memungkinkan adanya *resumable-process* dikarenakan proses pemindahan akan dilanjutkan ketika konektivitas terhubung kembali. Proses yang semacam ini hanya dimungkinkan apabila file yang dipindahkan di proses secara berkala atau tidak secara utuh. Dengan pemindahan secara berkala maka file tersebut haruslah dipecah menjadi beberapa potongan dan diproses secara bergantian. Konsep semacam ini di dalam beberapa kajian disebut sebagai *chunking* (Mostafa, et al, 2019).

Metode *chunking* itu sendiri adalah proses untuk memecah sebuah file yang utuh menjadi beberapa potongan dengan kapasitas dan ukuran yang sama. Hal ini dimaksudkan untuk membuat sebuah file dengan kapasitas yang besar menjadi beberapa potongan kecil dengan kapasitas file kecil. Sehingga dalam proses pemindahan file dari/ke *repository* memiliki beban yang kecil terhadap *server* (Ryan, et al, 2017). Juga tidak memerlukan penggunaan koneksi terhadap *http-port* yang terlalu lama sehingga putusnya koneksi dapat diminimalisir. Apabila meninjau dari konsep dasarnya, sangat layak jika metode *chunking* ini diaplikasikan dalam rancangan sistem yang bertujuan untuk memaksimalkan kinerja *server* dan mempercepat proses pemindahan file dari/ke *repository*. Terutama dengan resiko kegagalan pengunggahan file yang rendah. Sehingga metode *chunking* adalah modal untuk menjadi komponen penting dalam menciptakan sistem yang mampu mengatasi masalah yang ada. Pada dasarnya setiap pengiriman file, secara default telah dilakukan pemotongan file pada *layer* TCP (Transmission Control Protocol). Pemotongan ini juga dapat disebut dengan *chunking*, walaupun dengan beberpaa pendekatan yang berbeda. Sedangkan apabila dalam merancang sistem repository dalam basis web maka diperlukan *port* http/https dalam pengembangannya. Sehingga apabila memanfaatkan pemotongan file/data dalam layer TCP, maka tidak dapat dilakukan menipulasi atau scenario dalam menyelesaikan masalah. Mengingat tingkat kerumitan mendasar untuk memprogram pada *layer* tersebut dan pendekatan sistem yang mengarah pada basis web. Diperlukan pemotongan pada *port* http/https dimana berada satu *layer* di atas *layer* TCP. Apabila tidak memanfaatkan *port* http maka proses pengiriman file tidak dapat dilakukan

secara berkala dan hanya seperti sistem pengiriman konvensional lainya. Manipulasi atau pemograman yang memanfaatkan http/https dapat memungkinkan adanya pengiriman secara berkala. Dalam pengembangan sistem berbasis web maka digunakan *port* http/https untuk mengakses sistem yang dibuat, Pengguna sistem juga akan berkomunikasi dan menggunakan sistem melalui *port* http/https ini. Metode *chunking* yang diberlakukan dan bekerja pada *port* ini memungkinkan adanya manipulasi pengiriman file dari pengguna yang menggaksesnya. Apabila hanya memanfaatkan TCP saja maka file dari pengguna hanya dapat dimanipulasi secara lokal dan tidak dapat ditransfer ke dalam *database* secara online. Karena *database* dan *server* pada sistem berbasis web akan berkomunikasi pada *port* http/https. Maka dari itu tidak cukup memanfaatkan pemotongan file yang terjadi pada layer TCP karena fungsi dari sistem dan basis sistem yang diperuntukan dalam menangani masalah pada sistem berbasis web (Dildar, et al, ,2019).

Namun di dalam sebuah arsitektur sistem diperlukan beberapa komponen yang saling beradaptasi satu sama lain untuk membuat kinerja keseluruhan sistem maksimal. Metode *chunking* itu sendiri dalam prosesnya menghasilkan potongan-potongan data yang banyak, sehingga dibutuhkan penyimpanan data yang dapat mengelola potongan-potongan tersebut dan menggabungkannya dengan baik. Sehingga dibutuhkan media penyimpanan yang dapat menyimpan dan mengelola berdasarkan kebutuhan dari sistem. Hasil dari proses *chunking* tidak dapat disimpan dalam *file-system* ataupun direktori dari pengguna. Karena *chunking* mengakibatkan banyaknya potongan file yang harus disimpan. Maka dari itu *database* merupakan komponen yang harus digunakan sebagai media penyimpanan pada sistem. Database yang berkonsep *cloud* dapat berintegrasi secara online menggunakan internet. Rancangan sistem memerlukan komponen *database* yang mampu menjadi *repository* bagi file yang dihasilkan dari proses metode *chunking*. *Database* yang diperlukan haruslah mampu menangani file hasil dari proses *chunking* sebagai dokumen/file. Meninjau dari hal tersebut, *database* yang digunakan haruslah berjenis *unstructured database*. Karena *database* jenis ini memiliki fleksibilitas tinggi dan mampu mengolah file yang dihasilkan dari proses *chunking*. *Unstructured database* memiliki sifat *scheme-free* yang berarti mampu menyimpan dokumen/file tanpa mendefinisikan terlebih dahulu struktur dokumen yang bisa disimpan di dalamnya. Dokumen/file yang tersimpan dalam *database* jenis ini mampu diolah sebagai dokumen sehingga data file tidak terstruktur bisa diolah di dalamnya (Chieh, Yin, 2015).

Dalam menrancang sistem yang dapat digunakan dalam kondisi internet yang *unstable* dan *unreliable*. Maka sistem perlu skenario untuk menghadapi kasus-kasus yang merepresentasikan masalah yang diangkat. Skenario ini dapat berupa konektivitas yang terputus dalam proses pengunggahan file dari pengguna ke *database*. Maupun tingkat ketersediaan file yang diuji pada kasus database yang mengalami masalah dalam melayani *request* dari pengguna. Oleh sebab itu dipilihkan *unstructured-database* yang memungkinkan adanya replikasi file didalamnya. Replikasi ini dapat memungkinkan adanya *database* utama dan *database* cadangan, dimana *database* cadangan berisi replikasi dari isi *database* utama. Kemudian *database-database* tersebut dimanipulasi/diprogram untuk dapat melayani request dari pengguna. Sebagai contoh, apabila *database* utama mengalami kerusakan dan tidak dapat melayani *request* terhadap file di dalamnya (Chieh, Yin, 2015). Maka *database* cadangan yang juga memiliki file yang dibutuhkan akan secara otomatis melayani *request* tersebut. Dengan kata lain database cadangan tersebut menjadi *database* utama untuk sementara waktu hingga database utama dapat digunakan kembali. Inilah mengapa replikasi pada file di dalam database sangat penting untuk dilakukan. Dapat dikatakan bahwa replikasi file dapat meningkatkan kinerja *database* dan meningkatkan aspek *high-availability* terhadap *request* dari pengguna (Peter, et al., 2005), Rancangan pada sisi database ini terlihat tidak terlalu penting atau mendesak untuk dilakukan. Namun dalam skala besar dan pengembangan yang lebih luas, scenario semacam ini sangatlah berguna dan layak untuk di implementasikan. Pengguna sebagai pihak yang mempercayakan filenya untuk disimpan tidak mengetahui kinerja dan pengelolaan pada sisi *database*. Pembuktian dan transparansi terhadap file pengguna dapat dibuktikan dengan proses pengunduhan file tersebut. Apabila pengguna mendapati file tersebut utuh kembali sesuai dengan file miliknya, artinya pengguna dapat mempercaya kinerja sistem secara keseluruhan. Pada akhirnya skema perancangan dengan komponen yang dianalisis di atas akan mampu menyajikan arsitektur sistem yang menjawab masalah yang ada. Implementasi dari komponen-komponen tersebut akan menjadi sebuah gagasan dimana wujud pendekatannya terfokus pada masalah kondisi internet yang *unreliable* dan *unstable*.

1.2 Rumusan Masalah

Berdasarkan permasalah yang diuraikan latar belakang, rumusan masalah yang akan dihadapi dalam pembuatan tugas akhir adalah sebagai berikut:

a. Seberapa handal metode *chunking* dalam menyelesaikan masalah pengiriman file dari *database* & meminimalisir resiko kegagalan dalam kondisi internet yang tidak stabil?

b. Apakah *unstructured database* dapat menjadi pilihan untuk diimplementasikan ke dalam sistem dengan metode *chunking*?

1.3 Batasan Masalah

Penelitian ini memiliki beberapa batasan agar tidak menyimpang dari permasalahan di atas. Batasan masalah pada penelitian ini sebagai berikut:

c. Metode *Chunking* digunakan sebagai pemecah file sebelum diunggah ke *database*.
d. *Database* yang digunakan adalah *unstructured database*.
e. *Database* berlokasi jauh dari *client-side* dan *server-side*.
f. Pengujian sistem dilakukan dengan parameter waktu.
g. File yang diunggah ke dalam *database* adalah file yang memiliki format
h. Besar file yang digunakan dalam pengujian sebesar 10 MB dan 100 MB.
i. Kecepatan konektivitas menyesuaikan kecepatan internet yang digunakan sehari-hari.
j. Sistem diuji dengan memutuskan koneksi internet ketika proses unggah/unduh file berlangsung.
k. Fokus terhadap sistem dengan metode *chunking* dalam meminimalisir resiko kegagalan pengunggahan file dalam kondisi internet yang tidak stabil.

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah untuk mengkaji metode *chunking* yang diimplementasikan ke dalam sistem untuk menyediakan rancangan sistem yang dapat melayani dalam keadaan internet yang *unstable* dan *unreliable*. Menguji metode *chunking* dalam kemampuan pengunggahan file dengan resiko kegagalan yang rendah.

1.5 Manfaat Penelitian

1.5.1 Bagi Peneliti Lain

Penelitian ini diharapkan dapat membuka penelitian lain terkait dengan metode *chunking* maupun rancangan sistem yang dapat menyelesaikan masalah yang ada pada pengguna saat ini. Penelitian ini juga diharapkan dapat menjadi rujukan bagi penelitian lanjutan yang lebih mendalam, terutama pada bidang informatika.

1.5.2 Bagi Instansi Terkait

Instansi maupun lembaga peneliti yang terkait dengan pengembangan sistem informasi maupun jaringan dapat menjadikan penelitian ini sebagai rujukan dalam merancang sistem *repository* maupun sistem informasi. Dalam mengembangkan sistem yang lebih baik dan efisien guna memudahkan pengguna dalam pemanfaatan sistem tersebut.

1.5.3 Bagi Penulis

Mendapatkan wawasan baru dalam bidang sistem *repository* dengan mengedepankan kenyamanan pengguna secara umum. Penulis juga dapat mengetahui tahap-tahap dalam merancang sistem berdasarkan masalah yang ada mulai dari teori, metode, dan langkah penyelesaian. Juga memiliki wawasan baru dalam menganalisa hasil pengujian dilakukan langsung.

1.6 Metodologi Penelitian

Langkah-langkah yang diterapkan dalam penelitian ini agar mencapai tujuan yang diinginkan adalah sebagai berikut:

l. Tahap pengumpulan informasi

Pengumpulan informasi dengan membaca dan menganalisa literasi untuk mendapatkan cara pandang dan solusi dari peneliti lain terhadap masalah yang serupa. Tahap ini penting dalam memunculkan metode terbaik yang dapat menyelesaikan masalah yang diangkat. Pengumpulan informasi berguna dalam menentukan proses perancangan dan pengujian yang akan dilakukan agar metode yang dipilih benar-benar tepat.

m. Tahap Perancangan Sistem

Tahap ini dilakukan dalam mengumpulkan dan memetakan komponen yang akan digunakan dalam sistem untuk digambarkan dan dijabarkan. Komponen-komponen yang digunakan seperti metode, *database,* dan arsitektur sistem dianalisa untuk membuat gambaran proses kerja sistem dari awal hingga akhir.

n. Implementasi Sistem

Implementasi dilakukan dengan merancang sistem menggunakan komponen yang telah dianalisa sebelumnya. Tahap ini menghasilkan rancangan sistem yang utuh yang siap dijalankan dan diuji terhadap skenario-skenario yang telah ditentukan sebelumnya. Rancangan

ini dibuat menjadi sistem yang dapat bekerja secara sempurna tanpa eror di dalamnya, Dengan begitu pengujian terhadap komponen didalamnya dapat lebih maksimal. Metode *chunking* dan *database* yang diimplementasikan menghasilkan sistem yang sesuai dengan apa yang direncanakan pada tahap-tahap sebelumnya.

o. Pengujian Sistem

Sistem yang telah diimplementasikan dan berjalan tanpa error diuji dengan menjalankan skenario yang telah diidentifikasi sesuai dengan masalah yang diangkat pada penelitian ini. Rancangan sistem menggunakan metode *chunking* dibandingkan dengan sistem umum yang biasa digunakan, Perbandingan ini menggunakan parameter waktu tempuh dalam proses pengunggahan file. Tahap ini juga menguji kinerja sistem secara keseluruhan sehingga dapat diketahui layak atau tidak untuk menjadi solusi atas permasalahan yang ada. Tahap ini menentukan agar sistem yang telah dibuat sesuai dengan tujuan penelitian.

1.7 Sistematika Laporan

Dalam penyusunan tugas akhir ini, sistematika penulisan dibagi menjadi beberapa bab sebagai berikut:

BAB 1 PENDAHULUAN

Berisi latar belakang mengenai permasalah aktual yang mendasari penelitian, masalah yang dihadapi, dan solusi yang ditawarkan terhadap masalah yang ada. Berdasarkan latar belakang yang ada, kemudian disusun rumusan masalah sebagai dasar perencanaan penyelesaian masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.

BAB 2 LANDASAN TEORI

Berisi uraian literatur-literatur dan pembahasan komponen yang sesuai dengan topik penelitian sebagai dasar untuk melakukan penelitian terhadap masalah yang diangkat dan beberapa penelitian sejenis yang telah dilakukan. Teori-teori yang diuraikan dalam bab ini menggunakan jurnal, buku, dan artikel sebagai bahan referensi dengan topik yang meliputi metode *chunking*, *unstructured database*, dan load-balancing dalam *database*.

BAB 3 METODOLOGI PENELITIAN

Berisi tahapan dan hal-hal yang dibutuhkan dalam penelitian sebagai komponen untuk merancang solusi atas permasalahan yang diangkat. Bab ini berisi pengumpulan data dan informasi, perancangan sistem dan pengujian terhadap sistem secara menyeluruh maupun setiap bagiannya.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Bab ini berisi implementasi dan pengujian atas apa yang telah dirancang sebelumnya. Hal tersebut meliputi: proses kerja sistem, hasil pengujian sistem, analisa metode dan komponen yang digunakan dalam sistem. Pengujian dapat memperlihatkan hasil pengujian dan memunculkan kelebihan dan kelemahan sistem yang dirancang. Hasil pengujian pada bab ini berisi argumen tentang layak tidaknya rancangan sistem ini menjadi solusi atas permasalahan yang diangkat berdasarkan data dari hasil pengujian.

BAB 5 KESIMPULAN DAN SARAN

Berisi kesimpulan mengenai hasil pengujian dan analisa yang telah sesuai dengan tujuan penelitian dan tidak melebar. Serta saran yang dimunculkan agar penelitian ini dapat dilanjutkan pada tahap yang lebih mendalam dengan mengembangkan kelebihan dan mencari informasi lain untuk menyempurnakan kekurangan yang ada.

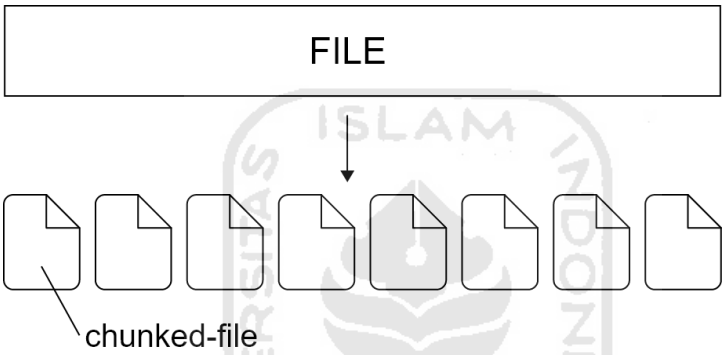
BAB II
LANDASAN TEORI

Dalam bab ini akan dijelaskan tentang dasar teori yang digunakan sebagai acuan dalam merancang dan mengembangakan arsitektur sistem. Dalam merancang sebuah sistem terdapat beberapa komponen yang digunakan baik itu metode, *database*, tampilan sistem, dan lain sebagainya. Seluruh komponen di dalam sebuah sistem harus dapat berinteraksi secara baik satu sama lain. Dalam bab ini antara lain akan membahas dan menjelaskan tentang metode *chunking*, *database* dan komponen lain dalam arsitektur sistem yang dibutuhkan. Seluruh komponen akan diimplementasikan ke dalam satu sistem yang dirancang sesuai dengan tujuan penelitian ini.

2.1 Metode *Chunking*

Metode *chunking* adalah proses untuk memecah/memotong file menjadi beberapa potongan dengan ukuran masing-masing sama. Metode ini dapat membuat satu buah file

dengan kapasitas yang besar menjadi beberapa potongan file dengan kapasitas kecil. Metode *chunking* dapat memungkinkan adanya pengiriman file secara berkala. Metode *chunking* bukan merupakan bahasa teknis atau nama paten dari sebuah metode melainkan istilah yang sering disebutkan dalam beberapa penelitian mengenai pemotongan/pemecahan file. Ukuran potongan file pada proses *chunking* dapat ditentukan oleh pengembang atau dengan menetapkan file yang akan diunggah dipotong menjadi 10 bagian. Proses penanganan terhadap file hasil metode *chunking* sangat beragam tergantung pemanfaatannya pada sistem. Fleksibilitas ini yang membuat metode *chunking* sering digunakan dalam beberapa penelitian terutama pada perancangan sistem *repository* (Ryan, et al., 2017).



Gambar 2. 1 Proses *chunking* menjadi 10 potongan.

Metode *chunking* membuat file yang utuh menjadi potongan-potongan file yang bersifat tidak terstruktur dan *unformatted*. Potongan file ini dapat dikelola menjadi satu file yang utuh apabila disimpan dan dikelompokan secara terurut. Karena dalam menangani *request* untuk unduh maka potongan-potongan file tersebut harus disatukan kembali agar menjadi file yang utuh. Proses pengiriman secara berkala yang dapat dilakukan membuat proses pengunggahan file menjadi lebih cepat dan terjamin. Dikarenakan adanya potensi untuk melanjutkan proses pengiriman pada potongan terakhir yang dikirim. Dalam istilah yang sering disebutkan dalam beberapa penelitian, proses metode *chunking* menghasilkan *chunked* file. Atau dalam istilah memiliki arti file yang telah terpotong, jenis file ini memerlukan penanganan khusus untuk dapat mengelolanya.

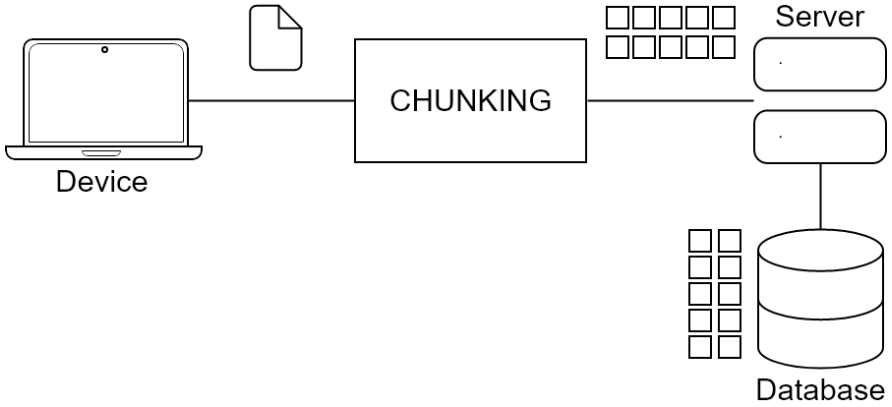
2.1.1 Chunked-file

Chunked file merupakan sebutan untuk file yang telah melalui proses *chunking*. File hasil metode *chunking* terdiri dari beberapa potongan dimana masing-masingnya memiliki format

dan ukuran yang sama. Namun format dari *chunked* file ini berbeda dengan file asli/induk sebelum melewati proses *chunking*. Hal ini dikarenakan hasil dari proses *chunking* adalah file tidak terstruktur atau dengan kata lain tidak berformat asli (Youjip, et al., 2014). Masing-masing potongan memiliki pembeda tergantung pendefinisian oleh pengembang, pembeda antara *chunkedfile* biasanya menggunakan *identifier-key* yang berupa *metadata*. *Identifier-key* berisi informasi yang bisa dimanipulasi berdasarkan urutan waktu tiba *chunked* file ke *database*, tergantung dari pengembang sistem yang akan mengatur file tersebut. Fungsinya untuk mengelola *chunked file* ketika terjadi *request* untuk menyatukan kembali dan melayani proses unduh. Tujuan dari pemotongan file pada sebuah sistem *repository* penyimpanan file adalah untuk memungkinkan adanya pengunggahan file secara berkala. Sehingga dapat dimanipulasi untuk adanya *resumable-upload* pada proses pengunggahan file.

2.2 Resumable File-Upload

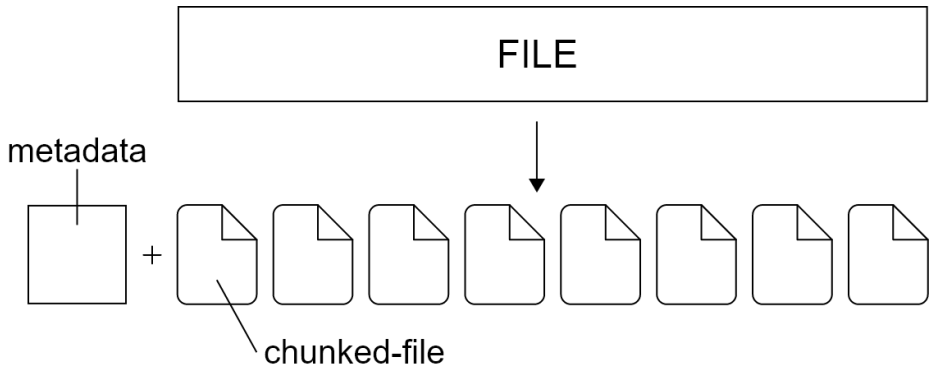
Resumable file-upload adalah proses yang memungkinkan adanya lanjutan dari proses *upload* file yang sebelumnya telah terhenti atau dihentikan. Proses ini banyak dijumpai dalam sistem yang menggunakan *cloud storage*, dimana sistem model ini sangat mengedepankan *high-availability* dan efektifitas sistem. Dewasa ini banyak pengembangan *framework* pemrograman yang berfokus pada *resumable-process* agar dapat diimplementasikan pada sistem. Mulai dari Resumable.js, Tus.io, Bootstrap Resumable-upload dan masih banyak lagi. Akan tetapi secara mendasar *resumable-process* dapat mungkin terjadi karena adanya pemecahan file pada proses pengunggahan. Pemecahan ini dilakukan untuk membuat proses *upload* dapat dilakukan secara berkala pecahan demi pecahan file. Pecahaan atau potongan file tersebut diberi identitas sebagai pembeda antara potongan file lainnya, identitas ini juga berfungsi sebagai sifat file. Agar ketika potongan-potongan file dimasukan ke dalam *database* dapat dikelompokkan berdasarkan file induknya/sebelum dipecah (Thanh, et al..2016). *Resumable-process* ini sangat penting untuk diimplementasikan kedalam sistem yang dapat menjadi solusi atas permasalahan konektivitas yang *unstable* dan *unreliable*.



Gambar 2. 2 Pengiriman file secara berkala

2.3 Identifier-key/Metadata

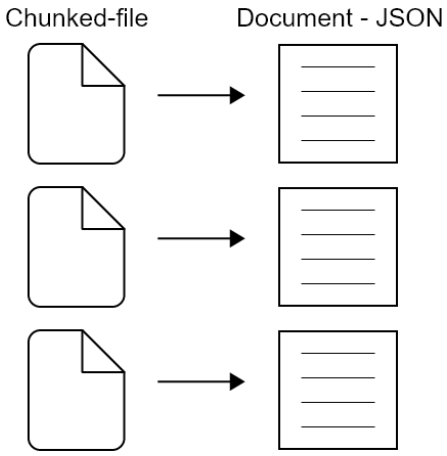
Identifier-key adalah kunci yang berisi informasi tambahan yang diberikan kepada setiap file agar dapat dibedakan satu sama lainnya. Informasi ini dapat mengelompokan suatu file yang memiliki sifat dan karakter yang sama, sekaligus memisahkan mereka agar dapat diidentifikasi. Pemberian informasi ini berfungsi untuk mengelola dan memanipulasi suatu file sesuai dengan kebutuhan sistem. Dalam beberapa penelitian, *identifier-key* ini dapat juga berupa *metadata* yaitu data yang berisi informasi suatu berkas. Informasi yang disematkan pada file dengan penamaan yang konstan atau tidak acak, karena fungsi dari *identifier-key* ini untuk memudahkan dalam pemanggilan file. Proses pemberian *identifier-key* pada suatu file pada umumnya terjadi pada *client-side* sebuah sistem. Untuk memastikan bahwa file tersebut telah secara baik dapat disimpan dalam *database* (Kei Ren, Garth. 2014). Sehingga fungsi *database* hanya menyimpan file tersebut sesuai dengan penamaan dan urutan file tersebut. Namun beberapa sistem juga meletakkan proses pemberian *identifier-key* ini pada *database*. Hal ini dimaksudkan untuk mempercepat proses pengunggahan file yang melibatkan pengguna dan proses pengiriman ke *database*. *Database* diberi tugas tambahan untuk memberikan *identifier-key* pada file yang datang untuk disimpan sesuai dengan informasi yang diberikan.



Gambar 2. 3 Proses pemberian *metadata*

24 Unstructured Database

Unstructured database adalah jenis *database* yang memiliki konsep NoSQL. *Database* ini juga dapat disebut sebagai sistem basis data *non-relational,* yang berarti sistem manajemen pada *database* jenis ini tidak mementingkan relasi antar tabel seperti pada *database* terstruktur (Chieh, et al..2015). Pengolahan data pada *database* ini dimuat dalam bentuk dokumen dan diolah dengan JSON. JSON merupakan format dokumen seperti halnya XML yang digunakan untuk berbagi data atau berkas. Dokumen JSON ini dapat diolah dan diprogram menggunakan *Javascript* yang membuat pengolahan *database* jenis ini lebih mudah karena setiap bahasa pemrograman mendukung pengolahan JSON. Sifat dokumen semacam ini memudahkan pengembang untuk mengubah atau mengembangkan *database* dan dokumen di dalamnya sesuai kebutuhan aplikasi (Mohammed, et al. 2017). Dalam *database* ini tidak diperlukan untuk membuat relasi antar tabel (*scheme-free*). Jika dalam *database* relasional pembuatan tabel dan kolom atau baris di dalamnya dihubungkan dan dibatasi dengan *foreign-key*/*primary-key*. Dalam *database* tidak terstruktur, menggunakan *collections* sebagai pengganti tabel dan penyimpanan file nantinya dilakukan berdasarkan *collection*. *Database* tidak terstruktur juga memiliki konsep *embedded-document* yaitu adanya baris yang tertanam di dalam kolom baris. Setiap barisnya memiliki jumlah kolom yang berbeda dengan baris lainnya. Berbeda dengan *relational database* yang berkonsep *fixed* atau memiliki jumlah baris yang sama pada setiap kolom (Erik, Zacharias. 2017). Desain fleksibilitas ini yang membuat *database* tidak terstruktur sering digunakan dalam menyimpan file semi-terstruktur maupun tidak-terstruktur.



Gambar 2. 4 Pengelolan file pada *unstructured database*

Unstructured-database dapat menyimpan file tidak terstruktur atau *unformatted* dikarenakan menangani file sebagai dokumen. Namun diperlukan beberapa tahap lagi untuk dapat menyimpan dan mengelola file tersebut secara baik. Seperti pemberian informasi terhadap file yang disimpan untuk mempermudah proses pemanggilan file. Banyak kelebihan dari menggunakan *unstructured database* terutama dapat dimungkinkan adanya replikasi *database* maupun replikasi file. Kedua replikasi ini memiliki banyak keuntungan terutama pada segi ketersediaan file terhadap *request* dari pengguna.

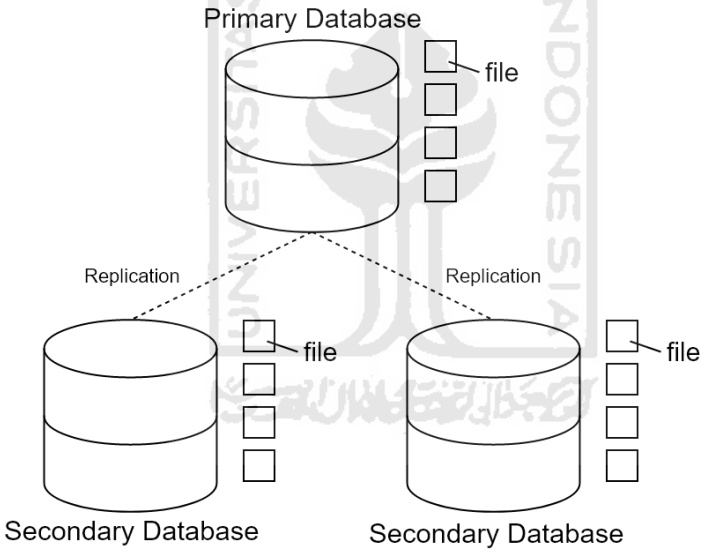
25 MongoDB

MongoDB adalah sistem basis data yang bersifat NoSQL dan berorientasi dokumen pada sistem penyimpanannya. *Database* NoSQL adalah basis data yang tidak menggunakan relasi antar tabel dan tidak menyimpan data dalam bentuk tabel dan kolom yang *fix*. MongoDB dikembangkan oleh tiga orang yaitu Kevin Ryan, Eliot Horowitz, dan Dwight Meriman dalam perusahaan MongoDB.inc. *Database* ini menawarkan aspek *high-availability* dan *high-performance* dalam penggunaannya. MongoDB dapat diakses melalui *command-line* dengan memanfaatkan beberapa perintah yang dibuat oleh MongoDB. Selain itu, MongoDB dapat juga diakses dalam versi *GUI* (*Graphical User Interface*) yaitu MongoDB Atlas. MongoDB Atlas adalah produk langsung dari MongoDB.inc yang juga dikembangkan bersamaan dengan MongoDB itu sendiri. MongoDB mengelola sistem penyimpanannya dengan *javascript*, dan menyimpan datanya pada sebagai dokumen yang terstruktur. *Database* ini juga masuk dalam kategori *unstructured database*, dengan aspek *high-availability* yang tinggi terhadap request dokumen di dalamnya. Salah satu praktek dari *high-availability* adalah tingkat ketersediaan file

tinggi dengan menyimpannya dalam beberapa *database*. Maksud dari beberapa *database* adalah replikasi *database* yang juga dapat mewujudkan replikasi file didalamnya.

26 Replikasi File

Replikasi file adalah sistem dinamik untuk membuat salinan sebuah file dengan identik. Mereplikasi file merupakan sebuah strategi untuk dapat mencapai *high-availability* file dan toleransi *error* yang tinggi. Hal ini dapat meningkatkan kinerja sebuah sistem karena adanya *load-sharing* dimana beban pelayanan dibagi dan didistribusikan ke beberapa *repository* penyimpanan file. Dengan begitu apabila *database* yang menyimpan file mengalami kerusakan maka sistem tetap dapat melayani dengan mengirim replikasi file dengan mengandalkan *sub-database* (Peter, et al., 2005). Adanya replikasi file didalam sebuah sistem *repository* dapat meminimalisasi waktu akses bagi pengguna dengan jumlah yang besar.



Gambar 2. 5 Proses terjadinya replikasi file

Proses replikasi dapat dilakukan dengan beberapa cara, salah satunya dengan file yang telah disimpan didalam *database* direplikasi dengan menyimpannya untuk kedua kali. Mengakibatkan terjadi proses penyimpanan sebanyak dua hingga lebih tahap. Juga dapat dilakukan replikasi file dengan melakukan replikasi pada *database* utama dan menyimpan isi dalam *database* tersebut. Dengan mereplikasi *database* maka sekaligus juga melakukan replikasi isi dalam *database* tersebut. Untuk memberi pembeda antara *database* utama dan *database* hasil replikasi maka diberi automasi untuk menjadikan *database* hasil replikasi sebagai *slave/secondary database.*

2.7 Secondary/Slave Database

Secondary atau *slave database* adalah *database* yang dibuat sebagai *backup* atau cadangan jika terjadi masalah terhadap *database* utama. Dalam industri *cloud computing* istilah *secondary database* sering juga disebut sebagai *slave database*. Prinsip kerja *secondary database* sama dengan *database* pada umumnya namun, fungsi utama dari *database* ini adalah *backup* atau cadangan ketika terjadi masalah pada *database* utama. *Database* ini biasanya berisi semua atau sebagian isi dari *database* utama, hanya saja prioritas penggunaan *database* ini dalam menangani *request* file berada dibawah *database* utama. *Secondary database* dapat dibuat dengan membuat beberapa *database* dan memberikan prioritas kepada setiap *database*, Atau dengan menggunakan beberapa fitur dara penyedia layanan *database* yang dapat dimanfaatkan dalam perancangan sistem. *Secondary database* harus dirancang secara otomatis untuk langsung mengambil alih *request* terhadap file didalamnya apabila *database* utama tidak mampu menangani *request*. Namun ketika *database* utama telah mampu menangani request terhadap file maka *secondary database* tidak lagi menangani *request* tersebut (Kalonji, et al., 2011). Hal-hal yang disebutkan di atas merupakan prinsip dan fungsi dasar dari *secondary database*. Tetapi dalam implementasinya banyak juga terdapat sistem yang memanfaatkan *secondary database* untuk membagi request terhadap file. Hal ini dimaksudkan untuk dapat memaksimalkan pelayanan terhadap permintaan pengunduhan file yang dapat ditangani oleh *database* utama dan *secondary database* secara bersamaan. Perancangan model seperti ini mengedepankan prinsip *high-availability* pada sektor penyimpanan file dan memaksimalkan kinerja *database* agar dapat selalu tersedia secara terus menerus. Pada akhirnya dengan memanfaatkan kinerja *secondary database/slave database* banyak sekali keuntungan yang akan diperoleh. Salah satunya adalah adanya penanganan *request* yang dapat dibagi. Beberapa komponen dibutuhkan untuk membuat sebuah sistem agar dapat berkomunikasi secara visual dengan baik oleh pengguna. Tampilan sistem diperlukan agar sistem dapat berintegrasi dengan baik dan interaksi dengan pengguna menjadi mudah.

2.8 Front-end

Front-end adalah tampilan atau antarmuka dari aplikasi yang dapat berinteraksi langsung dengan pengguna aplikasi. Antarmuka aplikasi ini mencakup hal-hal yang dapat dilihat langsung oleh pengguna seperti, warna, *font*, *layout website*, *grid*, gambar dan lain sebagainya. *Front-end* sangat berfokus pada estetika tampilan suatu aplikasi dan bertanggung jawab

terhadap kepuasan dan kemudahan pengguna dalam menggunakan aplikasi tersebut. Tampilan ini memungkinkan adanya interaksi antara pengguna dan sistem yang bertujuan untuk tercapainya keluaran yang diharapkan oleh pengguna. *Front-end* dari sebuah sistem pada umumnya memiliki logika dan penghubung untuk dapat berkoordinasi dengan *database* maupun *server*. Maka dibutuhkan *back-end* yang mengatur proses yang dikelola sesuai tujuan sistem.

2.9 Back-end

Back-end adalah logika dari suatu aplikasi yang menentukan alur dan performa dari aplikasi. *Back-end* meneruskan apa yang menjadi tujuan dari *front-end*, kemudian meneruskan data, input, dan perhitungan untuk dimanipulasi dan disimpan. *Back-end* bertanggung jawab dalam memberikan *output* yang berisi perhitungan atau apapun yang berhubungan dengan tujuan aplikasi. Dalam industri sistem informasi *back-end* juga berfungsi dalam menghubungkan tiga komponen yaitu *server, database* dan tampilan aplikasi. Ketiga komponen ini dihubungkan oleh *back-end* untuk mendapatkan output sesuai dengan model dari aplikasi yang dikembangkan.

2.10 Server-side

Server adalah program komputer yang ditujukan untuk melayani *service* tertentu dalam jaringan komputer. *Server* bekerja berdasarkan permintaan dari pengguna, dan diprogram untuk melayani secara otomatis. *Server* dibuat untuk merespon segala bentuk request dari pengguna secara cepat dan otomatis. Pada dasarnya, dalam sebuah program terdapat minimal 1 *server* yang menghubungkan pengguna dengan media penyimpanan. Berbeda dengan backend, *server* merupakan baris program yang bekerja tanpa diketahui oleh pengguna secara langsung. Namun keduanya sering dianggap “abu-abu” oleh banyak pengembang, dalam masalah istilah. Karena sebuah program dapat dikatakan sebagai *server* dan *backend* tergantung pendefinisian dari pengembang itu sendiri. Dalam beberapa kasus, sebuah program dapat disebut sebagai *server* apabila difungsikan sebagai pelayan dalam hal *request*. Berbeda dengan backend, bentuk pelayanan dari *server* ini tidak diketahui *output*-nya oleh pengguna. Ada banyak bahasa pemrograman yang dapat digunakan dalam membangun program *server*.

2.11 Golang

Golang atau dapat juga disebut bahasa pemrograman Go adalah bahasa pemrograman yang dikembangkan oleh Google. Golang ini banyak mengadopsi bahasa pemrograman C dan C++ dalam hal penulisan terstruktur dan pemrograman yang konkuren dan berurutan. Banyak pengembang yang menyamakan Golang sebagai bentuk baru dari bahasa C. Karena sintaks antara Golang dan C yang hampir mirip, dan disediakan secara *open-source*. Bahasa Golang telah mendukung *multitasking* atau pemrosesan data dalam banyak *processor* dalam waktu bersamaan. Golang banyak digunakan dalam lingkungan pengembangan aplikasi karena *package* yang lengkap dan dapat dikembangkan secara mandiri oleh pengembang aplikasi. Golang juga banyak dikembangkan untuk dapat berintegrasi dengan komponen lain dalam sistem. Pemanfaatannya juga beragam mulai dalam hal pengembangan *server* hingga perancangan *back-end*. Golang dapat dikoneksikan dengan *client-side* melalui banyak cara, salah satunya melalui *server* lokal seperti *localhost* dan lain sebagainya.

2.12 Localhost

Localhost adalah istilah *default* yang merujuk pada komputer sebagai *server* lokal. *Server* ini dapat diakses melalui *default IP address* yaitu 127.0.0.1 yang juga disebut sebagai *loopback address*. Dalam lingkungan *web development*, localhost ini dimanfaatkan pengembang dalam pengecekan situs yang dibuat. Localhost dimanfaatkan dalam menjalankan program yang telah dibuat untuk mengetahui *error* di dalamnya. Karena localhost merupakan *server* lokal maka tidak membutuhkan adanya koneksi internet. Pengembang dapat menjalankan *server* lokal ini pada banyak *port*, *port* itu sendiri adalah jalur komunikasi yang mendukung koneksi dalam jaringan. Sehingga pengembang dapat menjalankan banyak localhost dalam *port* yang berbeda-beda. Namun *port* ini hanya dapat digunakan oleh satu program saja, dan tidak dapat menggunakan *port-port* yang digunakan oleh komputer. Localhost dapat dijalankan dalam beberapa protool jaringan, salah satunya adalah pada http.

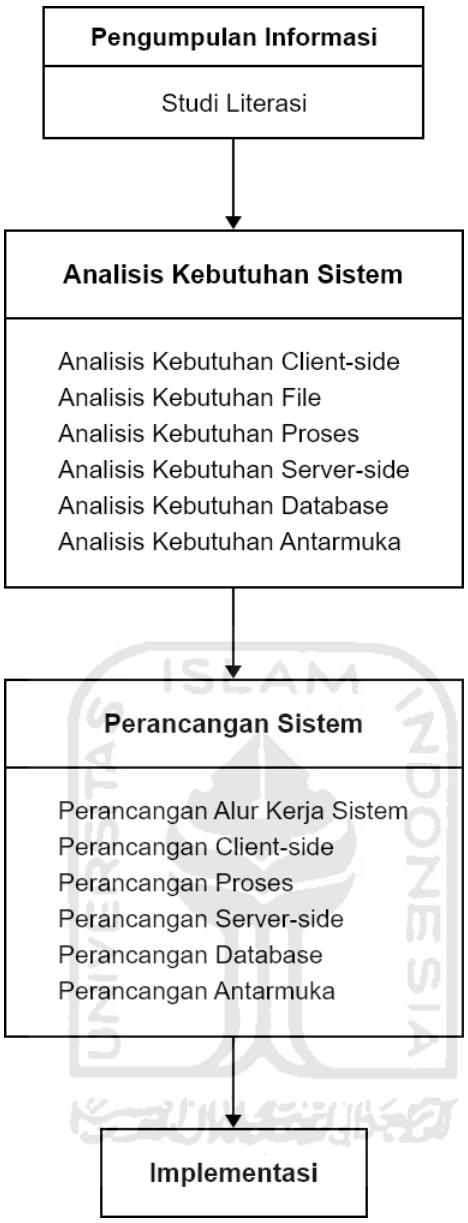
2.13 Http

Http atau *Hypertext Transfer Protocol* adalah protokol jaringan yang termasuk dalam lapisan *application layer*, yang difungsikan sebagai media pengiriman informasi. Informasi ini dapat berupa dokumen, file, gambar maupun video yang ditransfer antar komputer. Protokol ini mengatur format dalam transmisi data, dan mengatur bagaimana *server* memproses perintah yang masuk dari *client*. Http ini biasanya melayani sebuah perintah dengan membuat koneksi

port tertentu ke *server*. Dalam prakteknya, pengembang sering memanfaatkan http ini dalam proses pengembangan aplikasi. Protokol jaringan ini menjadi penghubung untuk melayani *client-side* dan *server* yang telah diprogram oleh pengembang aplikasi.

BAB III
METODOLOGI

Dalam merancang sistem *repository* dan penyimpanan file diperlukan komponen atau bagian yang harus berintegrasi dengan baik satu sama lain. Diperlukan analisis terhadap semua komponen yang dibutuhkan dalam merancang sistem pada bab ini. Bab ini akan menjelaskan tentang tahapan analisis setiap komponen untuk merancang sistem. Gambar 3.1 merupakan peta konsep untuk perancangan sistem.



Gambar 3. 1 Peta konsep dalam menganalisa sistem.

3.1 Studi Literasi/Pustaka

Tahapan ini merupakan pondasi atau dasar pengetahuan dalam perancangan sistem sebelum implementasi. Melalui studi literasi, sistem yang dirancang dapat sesuai dengan teori yang berkembang terutama seputar metode *chunking* dan rancangan *database*. Tujuan dari tahapan ini adalah mencari informasi dan pengetahuan mengenai sistem yang dapat digunakan dalam kondisi internet yang *unstable* dan *unreliable*. Ada banyak teori dalam merancang *database*, namun tidak semua teori dari setiap literasi bisa diimplementasikan ke dalam sistem pada penelitian ini. Karena beberapa rancangan, memiliki skema ataupun skenario yang tidak dapat berintegrasi maksimal dengan komponen lain dalam sebuah sistem. Studi literasi dapat

memunculkan kelebihan dan kelemahan setiap komponen sistem pada penelitian ini. Sehingga dapat mengetahui cara untuk memaksimalkan kelebihan dan meminimalisir kelemahan masing-masing komponen. Tahapan ini sangat berkaitan erat dengan algoritma, dengan algoritma maka sistem dapat bekerja secara benar sesuai kebutuhan. Metode *chunking* memerlukan algoritma yang matematis dan terstruktur dalam prosesnya. Bagaimana bisa memungkinkan adanya metode *chunking* dalam sistem apabila tidak mencari pengetahuan tentang algoritmanya. Sama halnya dengan rancangan *database*, ada beberapa jenis *database* yang sering digunakan dalam industri teknologi. Namun tidak semua *database* dapat digunakan sebagai media penyimpanan sistem dengan metode *chunking*. Maka tahap ini diperlukan peneliti dalam merencanakan komponen sistem yang dapat bekerja sesuai tujuan.

3.2 Analisis Kebutuhan Sistem

Dalam merancang sistem *repository* penyimpanan file, dibutuhkan komponen-komponen yang dapat berintegrasi dengan baik satu sama lain. Oleh karena itu diperlukan analisis kebutuhan sistem agar peneliti dapat menentukan apa saja komponen yang akan digunakan. Proses analisa ini dilandasi dengan pencarian informasi dan literasi pada tahap sebelumnya. Untuk menentukan tujuan dan skema sistem secara keseluruhan. Kebutuhan sistem ini dapat diketahui dengan menjabarkan bagian-bagian penting dalam satu sistem yang utuh. Sehingga bagian-bagian tersebut dapat diketahui fungsi dan proses yang terjadi didalamnya. Dengan begitu komponen yang akan digunakan dapat dianalisis sebelum diimplementasikan ke dalam sistem.

3.2.1 Analisis Kebutuhan *Client-side*

Tahapan analisis ini berguna untuk mendapatkan informasi tentang kebutuhan pada *client-side*. Dalam merancang sistem yang berinteraksi langsung dengan pengguna maka diperlukan analisis pada bagian *client-side*. *Client-side* merupakan bagian yang penting untuk dianalisis, bagian ini merupakan awal dari proses kerja sistem. Bagian ini berinteraksi dan berkomunikasi secara langsung dengan pengguna, agar sesuai dengan tujuan dari sistem. Interaksi ini bertujuan untuk meneruskan data/input yang dimasukan oleh pengguna, kemudian diteruskan dan dikelola pada *server-side*.

Terdapat 3 kebutuhan utama dalam *client-side* yaitu antarmuka, proses pemotongan file dan pemberian *metadata* pada *chunked-file*. Bagian antarmuka sistem haruslah mampu

menyediakan tampilan yang dapat berinteraksi dengan mudah. Karena hanya terdapat 2 proses pada sistem ini, maka antarmuka yang akan dibuat sebaiknya sesederhana mungkin. Proses *chunking*/pemotongan file terjadi pada bagian ini yaitu ketika file di unggah oleh pengguna. Sehingga diperlukan analisis untuk menangani proses pemotongan file sebelum mengirimkannya ke *database*. Setelah proses *chunking* berlangsung *client-side* masih bertugas untuk memberikan *identifier-key/metadata* pada setiap *chunked-file*. Proses ini sangat penting untuk dianalisis, karena berhubungan langsung dengan file yang diunggah pengguna. Tahapan ini juga harus menentukan parameter apa saja yang digunakan dalam *metadata* yang akan disematkan pada file.

Analisis pada tahap ini juga berhubungan dengan kebutuhan integrasi antara *server-side* dan *client-side* dalam menangani file yang akan dikirimkan. Integrasi ini menentukan kecepatan proses pengiriman file ke *database*, maka analisis ini sangat penting untuk dilakukan. Tahap ini juga menentukan apa saja yang akan ditampilkan dalam antarmuka.

3.2.1 Analisis Kebutuhan File

Tahapan ini menganalisis skenario untuk file yang akan diunggah dan diteruskan ke dalam *database*. Analisis ini penting dilakukan, agar dapat file yang diunggah dapat dikirim dan disimpan dengan baik nantinya. Karena perlu adanya informasi tentang berapa jumlah potongan file dalam satu file yang utuh, atau juga menentukan ukuran setiap potongan file tersebut. Setelah file tersebut bertransformasi menjadi *chunked-file* yang dilakukan pada *client-side*, proses akan ditangani oleh bagian *server-side*. Sehingga perlu analisis terhadap efektifitas penanganan file, agar dapat dikirim dengan baik dalam kondisi internet yang *unstable* dan *unreliable*. Pada proses pengiriman file, diperlukan juga skenario untuk menentukan proses pengiriman pada setiap potongan file tersebut. Penanganan file juga dilakukan dengan menganalisis metode pengiriman yang akan dilakukan. Karena diperlukan penanganan khusus terhadap *chunked-file*, karena jenis file nya yang tidak berformat dan sulit untuk diidentifikasi pada *database*. Fokus pada tahapan analisis ini adalah menentukan beberapa skenario terhadap file, yaitu sebagai berikut:

a. Jumlah potongan file untuk satu file yang akan diunggah.
b. Ukuran setiap *chunked-file*.
c. Pemberian identifikasi untuk setiap potongan file/*chunked-file*.
d. Jenis dan format file yang dapat dilakukan *chunking*.

e. Menangani file tidak berformat agar mudah dikelola oleh *database* nantinya.

3.2.2 Analisis Kebutuhan Proses

Tahapan analisis ini dilakukan untuk mengetahui bagaimana proses pengunggahan file berjalan. Ketika proses unggah/unduh file terjadi maka proses yang dibutuhkan sama seperti pada sistem konvensional lainnya. Pada dasarnya proses pengiriman dari *device* pengguna ke *database* ataupun pengiriman dari *database* ke *device* pengguna adalah sama. Karena proses tersebut hanya merupakan tahap pengiriman file dari *device* pengguna menuju *database*. Namun apabila dimungkinkan adanya *resumable-process*, maka diperlukan skema khusus untuk menangani file yang sedang dikirimkan. Analisis ini dilakukan untuk mengetahui skema seperti apa yang harus digunakan dalam memungkinkan adanya resumable-process.

3.2.3 Analisis Kebutuhan *Server-side*

Analisis pada tahap ini perlu dilakukan, karena *server-side* merupakan penentu sistem dapat bekerja sesuai tujuan atau tidak. *Server-side* dirancang untuk dapat meneruskan file yang telah dipotong agar dapat disimpan di *database*. Rancangan *server-side* pada sistem harus dapat menjadi penghubung antara *client-side* dan *database*. Karena pengguna tidak berinteraksi langsung dengan *server-side* dan pengguna tidak mengetahui penanganan *server-side* terhadap file yang diunggah. Oleh Karena itu bagian ini haruslah dapat berkoneksi dan berkomunikasi dengan *client-side* dan *database*. Bagian ini berinteraksi dengan *chunked-file* dan dapat memanipulasi file tersebut agar sesuai dengan apa yang dibutuhkan *database*. Karena *database* tidak mengetahui jenis file tersebut, nama file tersebut dan kelompok file tersebut. Maka *server-side* bertugas dalam menentukan kebutuhan itu agar dapat dikelola oleh *database* dengan baik. Cara penanganan terhadap *chunked-file* yang dikirimkan dari *client-side* memerlukan analisis mendalam.

Ada beberapa faktor yang menentukan *chunked-file* tersebut dapat disimpan dan dikelola dengan baik di *database*. Faktor-faktor tersebut menjadi kebutuhan pada bagian *server-side* ini, dan dapat dijabarkan sebagai berikut.

a. Adanya identitas yang unik untuk setiap *chunked-file*.
b. Menentukan parameter pada setiap identitas file.
c. Identitas harus dapat mengelompokan file yang akan disimpan.
d. *Server-side* dapat berintegrasi dengan *database*.

e. *Server-side* menjadi penghubung antara *client-side* dan *database*.

3.2.4 Analisis Kebutuhan *Database*

Database adalah bagian sistem yang bekerja pada tahap akhir, dengan fungsinya sebagai media penyimpanan. Tahap analisis ini dibutuhkan untuk mengetahui rancangan *database* yang tidak hanya dapat menyimpan, namun juga dapat mengelola file dengan baik. *Database* yang dibutuhkan harus berintegrasi dengan *server-side*, karena file dikirimkan melalui *server-side*. Analisis pada bagian ini penting dilakukan untuk merancang *database* agar dapat melayani request pengunduhan dan pengunggahan dengan *high-availability*. Dengan kata lain, cara untuk memaksimalkan kinerja *database* agar selalu tersedia untuk melayani *request* dari pengguna. Rancangan *database* ini perlu analisis mendalam, karena menentukan kinerja sistem secara menyeluruh. *Database* juga merupakan bagian yang menjadi fitur unggulan dari sistem pada penelitian ini. Ada beberapa fokus untuk mendapatkan informasi dari tahap analisis pada *database* yaitu:

a. Rancangan *database* yang *high-availability* terhadap file di dalamnya.
b. Menyimpan dan mengelola file hasil proses *chunking* dengan baik.
c. Tidak membatasi format file yang disimpan.

3.2.5 Analisis Kebutuhan Antarmuka

Tahap analisis antarmuka sangat diperlukan dalam menentukan tampilan sistem agar mudah dipahami oleh pengguna. Meskipun rancangan sistem pada penelitian ini tidak membutuhkan banyak hiasan atau gambar, namun penting untuk tetap memperhatikan antarmuka sistem. Karena antarmuka merupakan komponen sistem yang berinteraksi langsung dengan pengguna. Antarmuka dirancang untuk dapat mengarahkan pengguna untuk menggunakan sistem sesuai dengan tujuan/proses bisnis sebuah aplikasi. Dalam rancangan sistem pada penelitian ini dibutuhkan antarmuka yang sederhana untuk pengguna dapat mengunggah file. Pengguna tidak perlu mengetahui proses yang berjalan pada sistem, namun mengetahui bahwa file tersebut telah terunggah dengan baik. Sehingga kebutuhan antarmuka tersebut dapat dijabarkan menjadi berikut:

a. Antarmuka Pengunggahan File

Antarmuka ini merupakan tampilan utama dari sistem yang menunjukan bahwa pengguna dapat melakukan pengunggahan file. Antarmuka dibuat sesederhana mungkin, karena tujuan

dan proses dari sistem ini adalah *repository* atau media penyimpanan file. Dikarenakan sistem pada penelitian ini tidak memiliki proses bisnis yang panjang, maka antarmuka pada sistem hanya membutuhkan satu halaman. Antarmuka ini diharapkan agar pengguna mengetahui bahwa diharuskan untuk memilih file yang akan diunggah tanpa mengetahui proses yang terjadi.

b. Antarmuka Proses Unggah

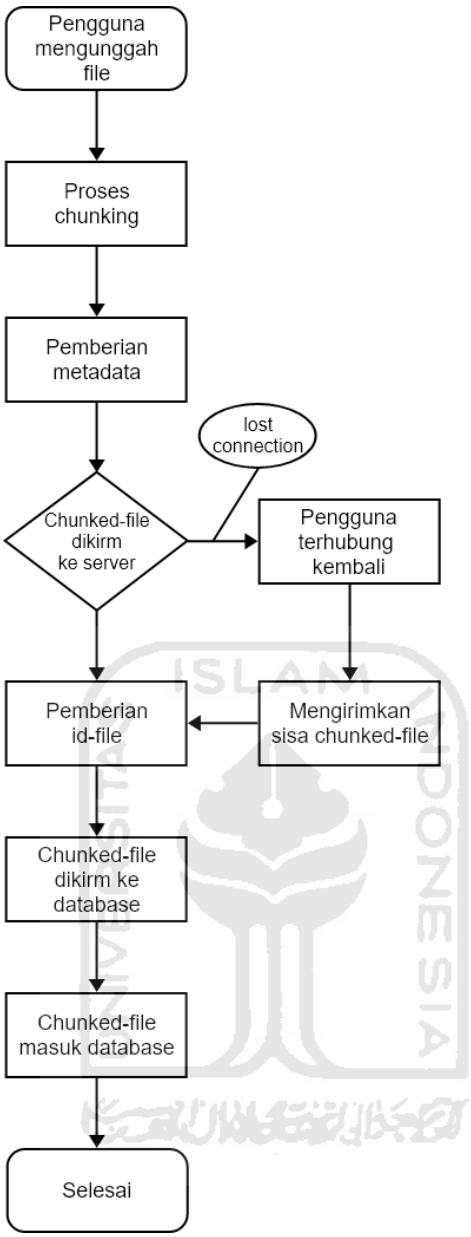
Antarmuka ketika proses pengunggahan berada satu *frame*/tampilan dengan antarmuka pengunggahan file, untuk menyederhanakan antarmuka sistem. Pada umumnya, ketika proses pengunggahan file terjadi maka ditampilkan sebuah *progress-bar* yang menunjukan persentase proses pengunggahan. *Progress-bar* ini direpresentasikan dalam bentuk persentase 0% hingga 100%. Antarmuka ini dirancang sesederhana mungkin agar pengguna dapat fokus memperhatikan file diunggah hingga 100%.

3.3 Perancangan Sistem

Setelah tahapan analisis kebutuhan sistem sebelumnya, kemudian dilakukan tahap perancangan sistem. Tahap perancangan sistem dilakukan agar setiap komponen pada sistem dapat dirancang untuk berintegrasi dengan baik sesuai kebutuhan. Setiap bagian dianalisis agar dapat mencapai tujuan sistem dalam menangani masalah internet yang *unstable* dan *unreliable*. Hal-hal yang telah dibahas pada analisis kebutuhan sistem menjadi bahan utama dalam perancangan sistem. Fokus utama dari tahap perancangan sistem ini adalah untuk menentukan rancangan yang sesuai dengan kebutuhan sistem. Kebutuhan sistem ini juga meliputi semua komponen didalamnya, mulai dari *client-side* hingga *database*.

3.3.1 Perancangan Alur Kerja Sistem

Dalam merancang sistem diperlukan konsep yang matang untuk mendapatkan rancangan yang sesuai demi mencapai tujuan sistem. Menentukan rancangan pada setiap bagian atau komponen didasari oleh tugas dan fungsi dari masing-masing komponen. Tugas dan fungsi setiap komponen ini menentukan kerja sistem secara keseluruhan. Oleh karena itu, diperlukan alur kerja yang menjadi struktur dari sistem secara keseluruhan. Alur kerja ini dirancang sesuai dengan kebutuhan sistem dalam mencapai tujuan yang telah direncanakan sejak awal penelitian. Gambar 3.2 dibawah ini adalah alur kerja sistem secara keseluruhan.



Gambar 3. 2 Alur kerja sistem

Penjelasan alur kerja sistem:

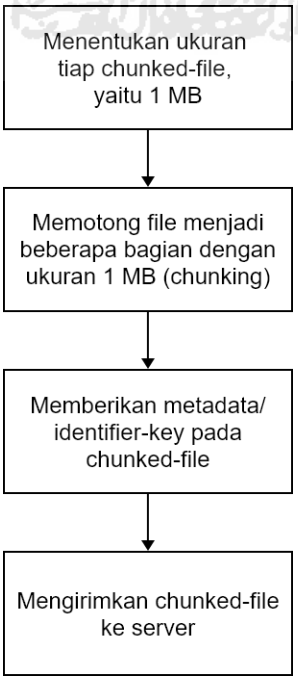
a. Mula-mula pengguna memilih file yang akan diunggah ke *database*, file ini berasal dari direktori lokal pengguna. Setelah file tersebut dipilih maka pengguna dapat melakukan pengunggahan file.
b. *Client-side* akan menampilkan progres dari proses pengunggahan file yang ditampilkan berdasarkan persentase 0-100%. Proses dari pemotongan dan pengunggahan file ini berlangsung pada *client-side*. Pada sistem ini, *client-side* merupakan bagian yang berinteraksi langsung dengan pengguna.

c. Ketika pengguna mengalami internet yang terputus saat proses pengunggahan file berlangsung. Maka secara otomatis sistem akan menghentikan proses pengunggahan file tersebut. Setelah internet dari pengguna terhubung kembali, sistem akan mulai memproses kembali. Sistem akan memproses pengiriman file dimulai dari *chunked-file* terakhir yang sebelum internet terputus. Dengan kata lain, sistem melanjutkan proses yang sempat terhenti karena internet yang terputus.
d. *Client-side* akan memberikan *metadata* pada setiap potongan file selama proses pengunggahan file berlangsung. Hal ini bertujuan agar *server-side* dapat mengidentifikasi dan mengelola *chunked-file* nantinya.
e. Kemudian *chunked-file* beserta *metadata/identifier-key* dikirimkan ke bagian *server-side* untuk dikelola kembali.
f. Setelah file tiba pada *server-side*, *chunked-file* ini akan dikelola untuk diberikan *id-file*. Proses ini berlangsung bersamaan dengan proses pengunggahan file. Pemberian *id-file* ini bertujuan agar *chunked-file* dapat disimpan dan dikelola dengan baik oleh *database* nantinya.
g. Ketika *chunked-file* telah ditangani dan dimanipulasi oleh *client-side* dan *server-side*, maka file siap dikirimkan ke *database*.
h. *Database* menyimpan file berdasarkan *id-file* dan mengelompokan *chunked-file* sesuai dengan *id-file* tersebut.

3.3.2 Perancangan *Client-side*

Dalam *client-side* terdapat 3 proses utama yang dikelola, yaitu proses pemotongan file, pemberian *metadata* , dan proses pengiriman file ke *server-side*. Pada *client-side*, algoritma dari metode *chunking* berjalan setelah pengguna men*submit* file yang akan diunggah. Setelah *chunking* dilakukan, *client-side* bertugas untuk mengirimkan/mengarahkan file menuju *database* melalui *server-side*. Dalam menangani file yang masuk diperlukan identifikasi ataupun ketentuan yang dibuat sebelumnya. Agar setiap file yang masuk kedalam sistem mendapat penanganan yang sama. Identifikasi ini berupa *metadata* yang berisi informasi untuk digunakan dalam mengidentifikasi *chunked-file*. Tahap perancangan sistem ini menentukan *metadata* berdasarkan beberapa parameter yaitu *key, filename, filesize, total chunks, dan file type*. Seluruh parameter ini dapat merepresentasikan file sesuai proses yang terjadi pada file tersebut. Informasi ini memudahkan dalam pengelolaan setiap file pada langkah selanjutnya termasuk dalam media penyimpanan atau *database*. Dengan begini seluruh file dengan format

apapun dapat diunggah pada sistem. File akan diperlakukan sama dan memungkinkan seluruh format file dapat diunggah melalui sistem. Dikarenakan *client-side* merupakan jalur masuk file kedalam sistem, maka ketentuan tersebut diberlakukan pada bagian ini. Berdasarkan tahap analisis kebutuhan sistem sebelumnya, algoritma *chunking* yang berjalan harus sama untuk setiap file yang masuk. Juga proses pengiriman dari file menuju *server-side* harus cepat dengan tingkat resiko kegagalan yang rendah. *Client-side* dirancang untuk dapat memotong file atau *chunking* menjadi beberapa bagian dengan kapasitas dan ukuran yang sama. Secara teknis, proses *chunking* terhadap satu buah file akan menjadi beberapa bagian *chunked-file*. Masing-masing *chunked-file* di program untuk memiliki ukuran atau kapasitas 1 MB. Ketentuan ini untuk memenuhi kebutuhan sistem agar file dapat ditangani dengan cara yang sama. Ukuran *chunked-file* ini ditentukan 1 MB, karena ukuran tersebut dapat mewakili 1% untuk satuan dan kelipatan 10. Alasan ini juga memudahkan dalam hal penelitian metode *chunking*, dimana proses dapat digambarkan dengan skala 1-100. Rancangan sistem ini ditujukan untuk mempercepat proses pengunggahan file dalam kondisi internet yang buruk. Maka apabila sistem memproses pengunggahan dan mengirimkan file berukuran 1 MB. Proses ini akan lebih baik dibandingkan membebani sistem dengan langsung mengirimkan ukuran file yang besar. 1 MB juga merupakan satuan terkecil dari angka 1 sampai 10. Dimana 10 MB masih merupakan ukuran file yang besar dan dapat membebani proses. Gambar 3.3 adalah proses yang terjadi pada *client-side*.

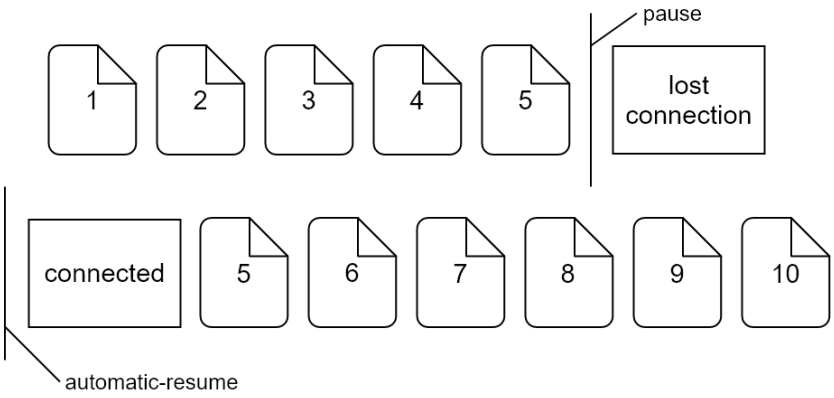


Gambar 3. 3 Proses yang terjadi pada *client-side.*

Client-side menentukan proses *chunking* dengan memotong file menjadi beberapa bagian dengan ukuran 1 MB. Setelah proses ini diimplementasikan, secara otomatis *client-side* akan melakukan pemotongan sesuai dengan ketentuan. Setiap *chunked-file* akan diberikan *metadata* yang berupa informasi dari file. Pemberian *metadata* ini bertujuan agar file dapat diidentifikasi dan dikelola oleh bagian setelahnya. *Client-side* juga memastikan koneksi yang baik dengan *server-side* karena banyak terjadi komunikasi antara 2 bagian ini. Koneksi ini merupakan jembatan agar file dapat tersampaikan dengan baik hingga tahap terakhir yaitu *database*.

3.3.3 Perancangan Skenario Proses

Perancangan proses ini merupakan tahap untuk merancang proses yang terjadi ketika pengunggahan/pengunduhan file terjadi. Dengan kata lain, bagian ini merupakan rancangan skenario untuk proses yang akan berjalan agar sesuai dengan kebutuhan sistem. Skenario ini dirancang berdasarkan kebutuhan sistem yang telah dianalisa sebelumnya. Berdasarkan kebutuhan sistem dan alur kerja sistem, dibutuhkan skenario untuk adanya proses *resumable-process.* Skenario *resumable-process* dibutuhkan ketika proses pengunggahan terhenti akibat masalah internet dari pengguna. Dengan begitu maka tahap proses di skenario untuk mengirim ulang *chunked-file* terakhir yang diunggah. Setelah *chunked-file* terakhir diunggah maka proses pengunggahan akan dilakukan hingga selesai. Sebagai contoh, apabila pengguna melakukan pengunggahan file dan berhenti pada *chunked-file* ke-5. Maka secara otomatis proses pengunggahan akan *pause* atau berhenti pada *chunked-file* terakhir. Setelah pengguna terhubung dengan internet, sistem secara otomatis akan melakukan *resume* atau melanjutkan proses unggah. Sistem akan mengunggah file kembali mulai dari *chunked-file* terakhir yang belum diunggah. Sistem melanjutkan proses unggah secara otomatis berdasarkan persentase dari *chunked-file* yang telah dikirimkan. Hal ini di skenario untuk menghindari terjadinya *file-corrupt pada file yang diunggah*. Karena pengunggahan file yang tidak menyeluruh atau hilangnya bagian file dalam proses pemindahan.

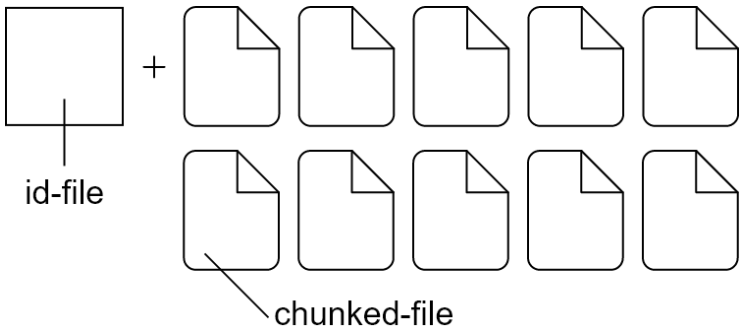


Gambar 3. 4 Skenario *resumable-process*.

Gambar 3.4 mengilustrasikan kasus ketika dalam proses pengunggahan file dengan jumlah *chunked-file* 10, kemudian proses terhenti pada *chunked-file* ke-5. Maka sistem secara otomatis akan mengirimkan kembali file mulai *chunked-file* ke-5 hingga selesai. Kemudian melanjutkan proses unggah file seperti biasa hingga seluruh *chunked-file* dari 1-10 telah terunggah 100%.

3.3.4 Perancangan *Server-side*

Berdasarkan analisis kebutuhan pada sistem, *server-side* bertugas untuk menerima *chunked-file* dari *client-side* dan mengirimkannya ke *database*. Namun sebelum menerima *chunked-file*, *database* membutuhkan adanya identitas untuk setiap *chunked-file.* Sehingga *server-side* dibutuhkan untuk melakukan hal tersebut, karena apabila beban tersebut diberikan kepada *database*. Maka *database* akan memiliki beban komputasi *database* yang besar dan membuatnya tidak dapat melayani *request* terhadap file dengan maksimal. *Server-side* sebagai jembatan antara *client-side* dan *database* harus dapat melakukan hal tersebut. Maka dirancang ketentuan pada bagian ini yaitu *id-file*. *Id-file* adalah data yang berisi informasi tentang sebuah file, berguna dalam memberi identitas terhadap file. *Server-side* bertugas memberikan *id-file* kepada setiap file yang dikirimkan ke *database*. Agar dapat disimpan dan dikelola secara berkelompok sesuai dengan file sebelum dipotong. Setelah itu diperlukan parameter untuk dapat mengelompokkan *chunked-file* secara baik dan terurut. Parameter ini juga dapat berfungsi sebagai pembeda *chunked-file* satu dengan lainnya. Parameter yang unik dalam kelompok *chunked-file* adalah waktu. Kelompok *chunked-file* memiliki kesamaan dari segala segi mulai dari format, jenis dan ukuran. Namun, waktu yang selesai untuk setiap *chunked-file* terunduh berbeda-beda. Parameter waktu tidak akan ada kesamaan antar *chunked-file* dalam satu kelompok. Karena proses pengiriman dilakukan secara berkala dan terurut dari urutan 1 ke 10.



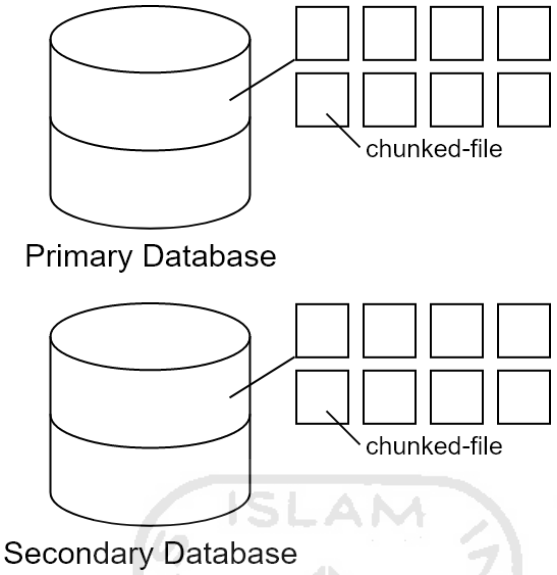
Gambar 3. 5 *id-file* pada kelompok *chunked-file*.

Secara umum *server-side* ini bertugas dalam menerima *chunked-file* beserta *metadata* didalamnya. Kemudian memberikan *id-file* kepada *chunked-file* tersebut berdasarkan parameter berupa *key* dan nomor dari setiap *chunked-file*. *Id-file* ini ditujukan untuk memberi pembeda diantara *chunked-file*, dan berguna dalam proses penyimpanan dalam *database*. Setelah proses tersebut dilakukan oleh *server-side*, kemudian file dikirimkan ke dalam *database* pada *collection* yang spesifik.

3.3.5 Perancangan *Database*

Ditinjau dari kebutuhan sistem secara keseluruhan, *database* adalah media penyimpanan yang harus mengelola *chunked-file*. Dalam penelitian ini digunakan jenis *database unstructured*, karena sifatnya yang fleksibel dan dapat menangani file tidak berformat. *High-availability* menjadi syarat utama rancangan *database* dalam sistem ini. *Database* ini melayani *request* dengan maksimal menggunakan sumber daya yang ada. Memaksimalkan kinerja *database* dengan mendesain *database* agar dapat melayani *request* tanpa adanya sumber daya yang mahal dan berlebihan. Salah satunya caranya adalah dengan memanfaatkan replikasi *database*. Replikasi membuat *database* utama dapat bekerja secara bergantian maupun bersamaan dengan *database* replikasi. Walaupun biasanya hasil dari replikasi *database* hanya digunakan dalam melayani ketika *database* utama mengalami kerusakan. Namun dengan memanfaatkan sumber daya ini, dapat membantu kinerja sistem secara keseluruhan terutama bagian ketersediaan file. Isi dari p*rimary database* yang telah berisi *chunked-file* beserta *metadata* dan *id-file* direplikasi ke dalam *slave/secondary database*. Dengan begitu *request* terhadap ketersediaan file dapat dilayani secara bergantian maupun bersamaan, antara *primary database* dan *slave*-nya. File dari *database* replikasi ini memiliki format dan isi yang sama seperti file yang dikirimkan dari *server-side*. Dalam penelitian ini *database* utama direplikasi

menjadi 1 atau 2 *slave database*. Dengan kata lain sistem pada penelitian ini memiliki 3 *database* untuk melayani *request* dalam melayani ketersediaan file.



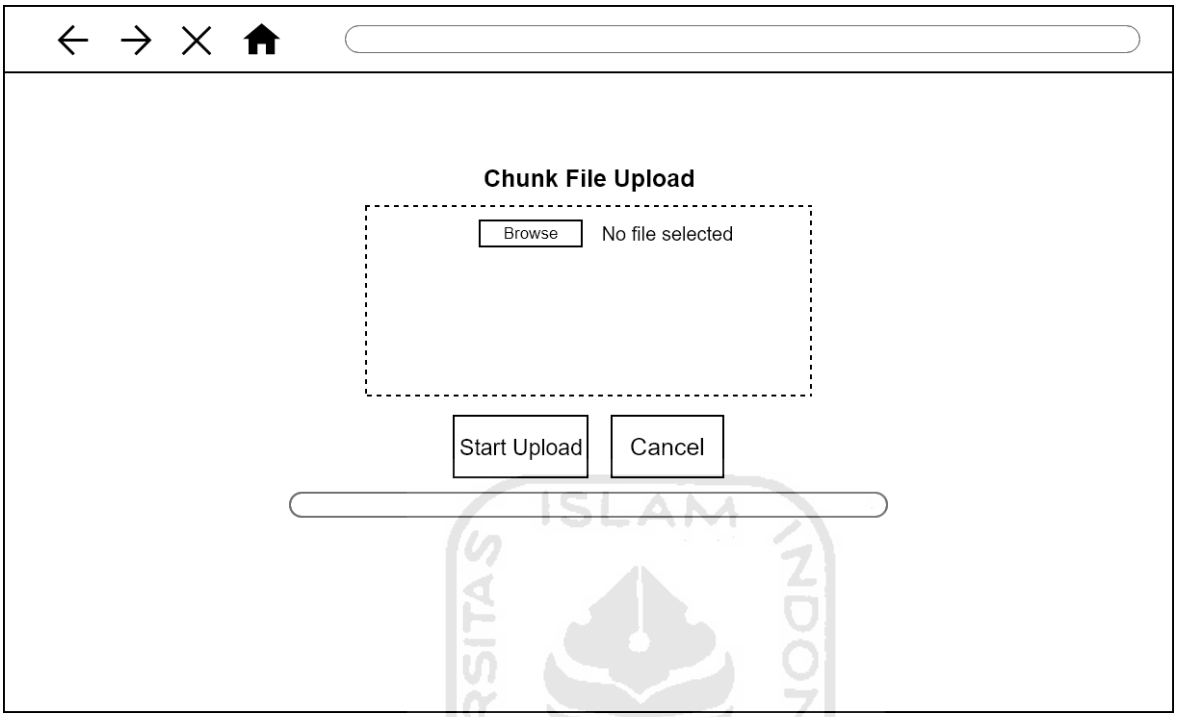
Gambar 3. 6 *Primary database* dan *secondary database.*

Dalam penggunaan *unstructured database*, dapat dimungkinkan replikasi dengan memanfaatkan beberapa fitur dari *platform* penyedia *database*. Dengan memanfaatkan fitur tersebut, *database* dapat diatur dengan memiliki beberapa *secondary database* di dalamnya. Sehingga *server-side* dapat mengarahkan *chunked-file* ke dalam *database* secara spesifik. Maksud dari spesifik adalah dengan memilih *database* mana yang akan menyimpan *chunked-file* tersebut. Apakah *database* utama atau *database* cadangan, semuanya dapat diatur sesuai kebutuhan sistem pada penelitian ini. Bahkan ketiga *database* sekaligus dapat menyimpan *chunked-file* dengan penanganan yang sama. Bagian-bagian di dalam sistem telah dirancang sesuai dengan analisis kebutuhan sistem. Namun ada bagian yang juga merupakan aspek penting dari sebuah sistem, yaitu antarmuka. Dimana bagian tersebut menjadi bagian yang berinteraksi secara langsung dengan pengguna sistem nantinya.

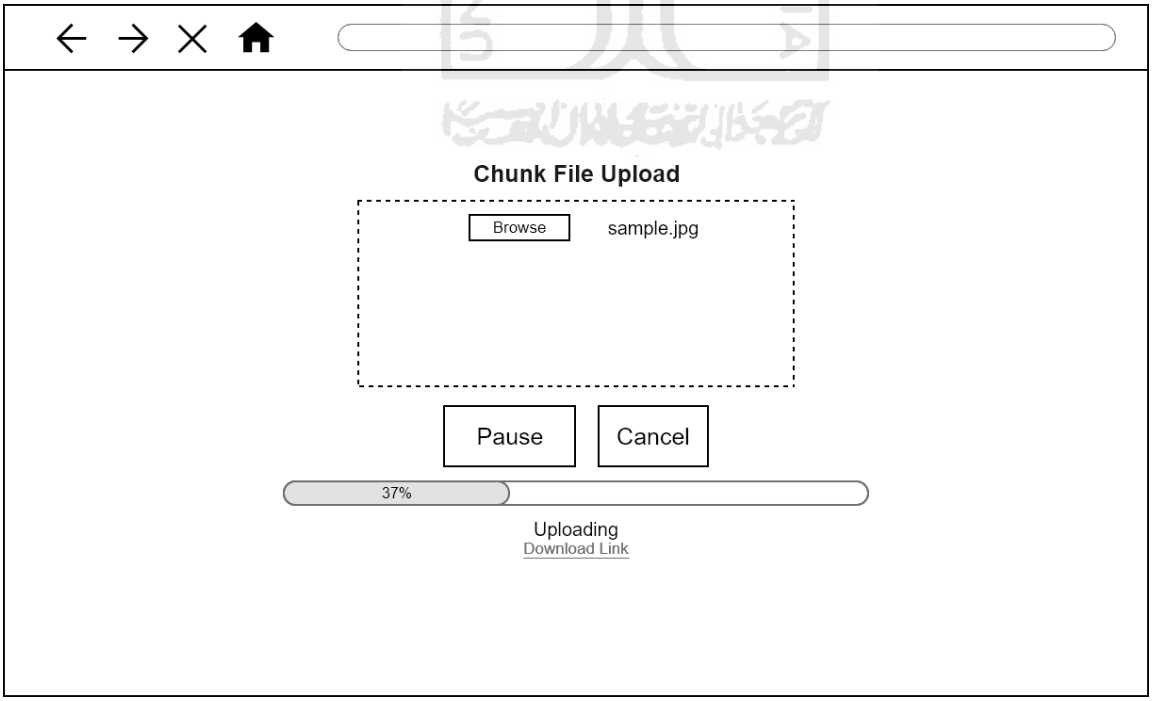
3.3.6 Perancangan Antarmuka

Perancangan antarmuka dilakukan dengan membuat *wireframe* terlebih dahulu. *Wireframe* ini bertujuan untuk membuat kerangka sistem. *Wireframe* sistem ditentukan berdasarkan kebutuhan sistem dam proses apa saja yang akan dilakukan oleh sistem. Karena sistem ini

merupakan sistem untuk pengunggahan file, maka dibuat karangka untuk halaman pengunggahan file. Gambar 3.7 dan 3.8 merupakan kerangka tampilan sistem.



Gambar 3. 7 *Wireframe* tampilan awal sistem.



Gambar 3. 8 *Wireframe* tampilan peunggahan file.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

Implementasi dilakukan berdasarkan analisis dan tahap perancangan yang telah dilakukan sebelumnya, agar sesuai dengan tujuan dari sistem pada penelitian ini. Perancangan pada bab sebelumnya telah menentukan apa saja yang harus dilakukan pada setiap komponen. Bagian implementasi ini menjelaskan rancangan dan teknologi yang digunakan untuk setiap komponen pada sistem. Teknologi yang digunakan sesuai dengan fungsi dan tujuan setiap komponen.

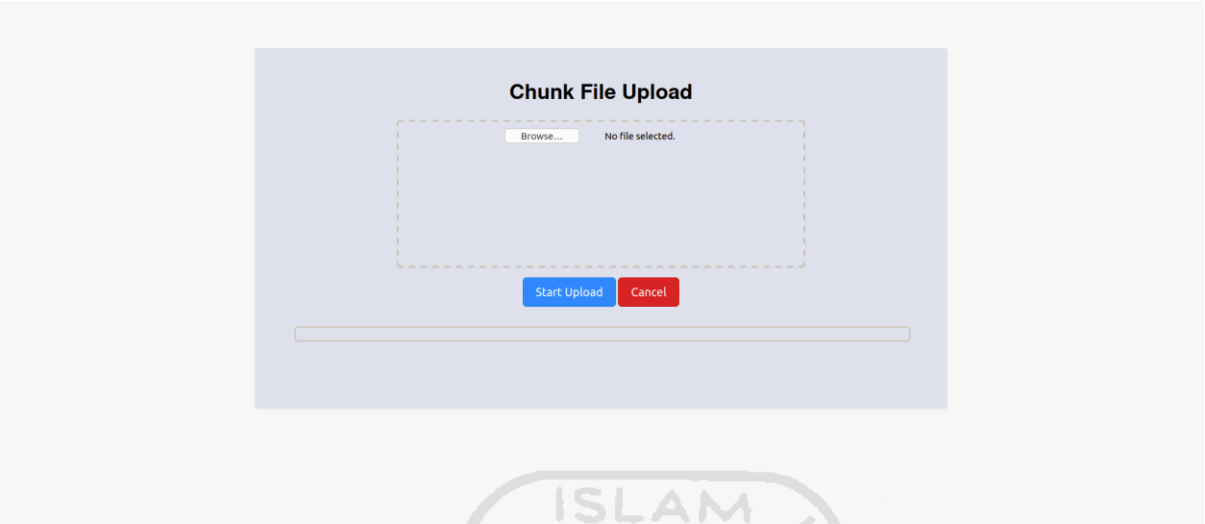
4.1 Client-side

Client-side memiliki 3 proses utama yang harus dijalankan sesuai dengan tujuan sistem. Bagian ini juga memiliki peran untuk berinteraksi langsung dengan pengguna. Interaksi ini merupakan alur awal untuk menentukan kinerja sistem. Pengguna berinteraksi dengan *client-side* melalui antarmuka sistem. Antarmuka ini ditujukan untuk mengarahkan pengguna dalam mengunggah dan mengunduh file. Dengan kata lain, sistem bekerja sesuai dengan apa yang dilakukan pengguna dengan *client-side* ini. Antarmuka ini adalah bagian dari *client-side* yang dapat diketahui oleh pengguna. Namun, ada 2 proses lagi yang tidak dapat diketahui oleh pengguna. Proses ini bekerja tanpa diketahui oleh pengguna dan menentukan file dari pengguna tersebut berhasil disimpan atau tidak. Kedua proses tersebut adalah proses *chunking* dan penyematan *metadata*. Kedua proses ini dirancang dalam bentuk program yang dituliskan dalam *javascript*. Proses dan komputasi yang bekerja dalam *client-side* akan dijelaskan secara rinci pada sub-bab ini.

4.1.1 Antarmuka

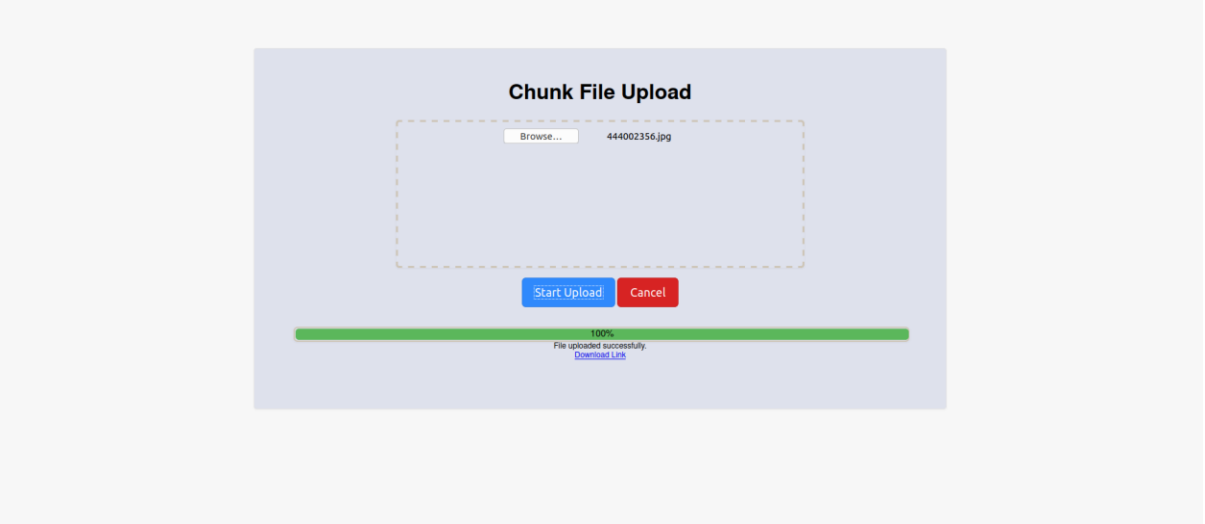
Antarmuka sering juga disebut sebagai *front-end*, yang merupakan bagian dari sistem yang berinteraksi langsung dengan pengguna. Sistem pada penelitian ini memiliki 2 tugas utama yaitu pengunggahan dan pengunduhan file. Antarmuka dirancang sesederhana mungkin karena sistem tidak memiliki banyak fitur. Sehingga sistem hanya memiliki 1 tampilan utama yaitu tampilan pengunggahan file. Seperti yang telah dibahas dalam tahap analisis kebutuhan sistem, antarmuka harus menampilkan *progress-bar* dan persentase proses pengunggahan. Antarmuka dibuat sesuai dengan kebutuhan, agar tercapainya tujuan dari sistem ini dibuat. Sistem tidak menampilkan gambar, ikon, atau tulisan yang berlebihan. Karena akan mengakibatkan besarnya *bandwidth* yang dibutuhkan untuk memuat tampilan. Mengingat sistem ini dirancang

untuk menyelesaikan masalah internet yang tidak memadai. Gambar 4.1 di bawah ini adalah antarmuka sistem yang akan digunakan.



Gambar 4. 1 Tampilan awal sistem.

Antarmuka utama sistem memuat judul, sebagai representasi proses yang akan dilakukan oleh sistem. Setelah judul dimuat fitur untuk mencari file yang akan diunggah pengguna, file ini akan didapat dari direktori lokal penggunga. Hal yang terpenting dari antarmuka ini adalah tombol, yang menunjukan bahwa pengguna akan mengirimkan file. Antarmuka juga memuat *progress-bar* yang akan menunjukan progres dari proses pengunggahan file. Seperti umumnya, *progress-bar* ini akan menunjukan persentase 0 hingga 100%. Ketika pengguna telah memilih file yang akan diunggah dan proses pengunggahan terjadi. Antarmuka akan menampilkan beberapa *output* sebagai indikasi proses yang terjadi. Gambar 4.2 adalah tampilan ketika proses pengunggahan terjadi.



Gambar 4. 2 Tampilan proses pengunggahan file

Ketika proses pengunggahan file selesai, antarmuka akan memuat tulisan "File Uploaded Successfully". Sebagai indikasi bahwa file dari pengguna telah disimpan dalam *database*. *Output* dalam tulisan ini selaras dengan *progress-bar* yang menunjukan persentase pada angka 100% yang berarti bahwa keseluruhan file telah terkirim. Antarmuka juga akan memuat link bertuliskan "*Download* Link", sebagai *url-link* untuk mengunduh file pengguna dari *database*. *Url* ini akan mengarah pada file yang telah berhasil diunggah sebelumnya. Setiap antarmuka dan *output* ini dikelola oleh sistem berdasarkan proses yang akan terjadi. Penjelasan tentang bagaimana proses yang terjadi di belakangnya akan dijelaskan dalam sub-bab ini.

4.1.2 Proses Persiapan *Chunking*

Chunking merupakan proses pemotongan file menjadi beberapa potongan dengan ukuran yang sama. Pada sistem ini, *chunking* akan menghasilkan beberapa *chunked-file* dengan ukuran 1 MB (*Megabyte*). Jumlah dari *chunked-file* ini akan tergantung dari ukuran file yang akan dikirimkan. Semakin besar ukuran file yang akan diunggah, maka semakin banyak jumlah potongan file tersebut. Oleh karena itu, pada *client-side* ini dideklarasikan fungi untuk menyimpan pengaturan tentang besaran *chunked-file* yang akan digunakan dalam sistem. Gambar 4.3 menunjukan kode program untuk mengatur ukuran dari *chunked-file*.

```
Function bigUpload () {
   this.settings = {
//Ukuran chunked-file dalam (bytes)
       'chunkSize': 1000000,
//Maksimal file (2GB)
      'maxFileSize': 2147483648,
};
```

Gambar 4. 3 Kode program persiapan *chunking*.

Gambar 4.3 merupakan pengaturan untuk menentukan ukuran *chunked-file* disimpan dalam fungsi *bigUpload*, seperti yang tertulis pada baris ke-1. *Chunked-file* ditentukan untuk berukuran 1.000.000 *bytes* atau sama dengan 1 *Megabyte*. Dalam kode program pada gambar 4.3 tersebut juga diatur mengenai ukuran file maksimal yang bisa diunggah yaitu sebesar 2 Gb. Pendeklarasian ini merupakan aturan pertama dari rancangan sistem, dan akan digunakan pada proses-proses selanjutnya. Setelah ukuran dari *chunked-file* ditetapkan, dilakukan beberapa persiapan sebelum memproses file yang akan dipotong nantinya. Gambar 4.4 adalah kode program untuk mempersiapkan pemrosesan file.

```
This.uploadData = {
       'uploadStarted': false,
       'file': false,
       'numberOfChunks': 0,
       'aborted': false,
       'paused': false,
       'pauseChunk': 0,
       'key': 0,
       'timeStart': 0,
       'totalTime': 0
   };
this.resetKey = function() {
          this.uploadData = {
              'uploadStarted': false,
              'file': false,
              'numberOfChunks': 0,
              'aborted': false,
              'paused': false,
              'pauseChunk': 0,
              'key': 0,
              'timeStart': 0,
              'totalTime': 0
          };
       };
```

Gambar 4. 4 variabel dalam proses *chunking*

Persiapan ini dilakukan untuk mengatur nilai dari setiap variabel yang akan digunakan baik dalam pemrosesan file maupun pengunggahan file. Variabel-variabel ini akan digunakan selama proses pengiriman file terjadi pada *client-side*. Pada kode program terdapat fungsi

"*resetKey*" dengan nilai dari setiap variabel didalamnya adalah 0 dan *false*. Kode program ini dimaksudkan agar sistem mereset setiap variabel sebelum adanya proses pengunggahan baru terjadi. Dengan kata lain, membersihkan nilai dari setiap variabel untuk diisi dengan nilai baru dari pemrosesan file baru setelahnya. Setelah pendefinisian variabel ini, dilakukan pemrosesan sebelum file diunggah. Pemrosesan ini berupa pembacaan beberapa informasi dari file yang akan diunggah. Dalam tahap pemrosesan, dilakukan juga penghitungan jumlah *chunked-file* untuk 1 file yang akan diunggah. Gambar 4.5 adalah kode program untuk melakukan pemrosesan file sebelum dikirimkan.

```
    This.processFiles = function() {
        this.resetKey();
        this.uploadData.uploadStarted = true;

        this.$(this.settings.progressBarField).style.backgroundColor           =
this.settings.progressBarColor;
        this.$(this.settings.responseField).textContent = 'Uploading…';
        this.$(this.settings.submitButton).value = 'Pause';

        this.uploadData.file = this.$(this.settings.inputField).files[0];

        var fileSize = this.uploadData.file.size;
        if(fileSize > this.settings.maxFileSize) {
            this.printResponse('The file you have chosen is too large.', true);
            return;
        }

        this.uploadData.numberOfChunks       =       Math.ceil(fileSize       /
this.settings.chunkSize);

        this.sendFile(0);
    };
```

Gambar 4. 5 Kode program pemrosesan file.

Kode program gambar 4.5 berfungsi untuk melakukan pemrosesan pada file yang akan diunggah nantinya. Kode program ini bekerja dengan menghitung ukuran dari file yang akan diunggah. Namun proses menghitung. sistem akan melakukan reset dengan mengosongkan nilai dari setiap variabel pada fungsi "*resetKey*". Kemudian sistem melakukan pengecekan, terkait apakah file yang diunggah pengguna lebih besar dari batas yang ditentukan sistem. Jika file tersebut lebih besar dari batas yang ditentukan, sistem akan menyatakan bahwa "*The file you have chosen is too large*". Setelah itu sistem akan melakukan perhitungan jumlah *chunked-file* untuk 1 file yang akan diunggah. Perhitungan ini dilakukan sistem dengan membagi ukuran file dengan ukuran *chunked-file*, dimana ukuran *chunked-file* ini adalah 1 MB.

4.1.4 *File Upload* Menggunakan Metode *Chunking*

Pengunggahan file menggunakan metode *chunking* dilakukan pada bagian *client-side*. Prinsip utama dari metode pengiriman ini adalah mengirimkan file berupa potongan-potongan file. Gambar 4.6 merupakan kode program untuk proses pengiriman dengan metode *chunking*.

```
    this.sendFile = function (chunk) {
        this.uploadData.timeStart = new Date().getTime();
        if(this.uploadData.paused === true) {
            this.uploadData.pauseChunk = chunk;
            this.printResponse('Upload paused.', false);
            return;
        }
        var start = chunk * this.settings.chunkSize;
        var stop = start + this.settings.chunkSize;
        var reader = new FileReader();
        reader.onloadend = function(evt) {
            xhr = new XMLHttpRequest();
            console.log(chunk);
            console.log(parent.uploadData.key);
            console.log(parent.settings.scriptPathParams);
            xhr.open("POST", parent.settings.scriptPath + '/upload/' +
                        parent.uploadData.key +
                        '/' + chunk +
                        "?filename=" + parent.uploadData.file.name +
                        "&filesize=" + parent.uploadData.file.size +
                        "&mimetype=" + parent.uploadData.file.type +
                        "&totalchunk="  +  parent.uploadData.numberOfChunks,
true);
            xhr.setRequestHeader("Content-type",     "application/x-www-form-
urlencoded");
            xhr.onreadystatechange = function() {
                if(xhr.readyState == 4) {
                    var response = JSON.parse(xhr.response);
                    if(response.errorStatus !== 0 || xhr.status != 200) {
                        parent.printResponse(response.errorText, true);
                        return;
                    }
                    if(chunk === 0 || parent.uploadData.key === 0) {
                        parent.uploadData.key = response.key;
                        parent.$(parent.settings.resultLink).innerHTML   =   "<a
href=\"" + parent.settings.scriptPath + "/download/" + parent.uploadData.key
+ "\"> Download Link </a>";
                    }
                    if(chunk < parent.uploadData.numberOfChunks) {
                        parent.progressUpdate(chunk + 1);
                        parent.sendFile(chunk + 1);
                    }
                    else {
                        parent.sendFileData();
                    }
                }
            };
            xhr.send(blob);
        };
```

Gambar 4. 6 Proses pengiriman file

Setelah sebelumnya telah dijelaskan tentang pemrosesan terhadap file yang akan diunggah. Kemudian pada gambar 4.6 adalah tahap untuk melakukan pengiriman file menggunakan metode *chunking*. Sebelumnya sistem akan melakukan pengaturan waktu dengan mengatur waktu sejak pengunggahan dimulai, dan menghitung sisa waktunya hingga pengunggahan selesai. Kemudian dilakukan pengecekan, terkait apakah proses dibatalkan atau di*pause* oleh pengguna. Jika proses dibatalkan maka sistem tidak akan memproses file untuk diunggah dan jika di *pause*, sistem akan menyatakan bahwa "*Upload paused*". Setelah itu sistem akan mengirimkan *metadata* yang berisi informasi dari file yang diunggah. Apabila *chunked-file* pertama telah dikirimkan, sistem akan menyediakan link untuk langsung mengunduh file tersebut. File tersebut berasal dari *link* "*Download Link*" yang berasal dari halaman pada localhost:8000. Kemudian selama proses berlangsung file akan dipotong menjadi *chunked-file* sesuai ketetapan sistem sebelumnya. Gambar 4.7 adalah proses *slice* atau memotong file.

```
var isIE = ((navigator.userAgent.indexOf("MSIE") != -1) ||
(navigator.userAgent.indexOf("Trident") != -1));
var blob = this.uploadData.file.slice(start, stop);
       if(isIE){
          reader.readAsArrayBuffer(blob);
       } else {
          reader.readAsBinaryString(blob);
       }
    };
```

Gambar 4. 7 Proses slice file

Sebelum memotong file menjadi *chunked-file* yang ditentukan, sistem akan melakukan pengecekan terkait aplikasi *web-browser* yang digunakan. Apabila *web-browser* yang digunakan adalah Internet Explorer, dimana tidak memiliki *method* seperti "readAsBinaryString". Maka digunakan *method* yang lain yaitu "readAsArrayBuffer". Kedua *method* ini digunakan agar *web-browser* dapat membaca isi dari file yang diunggah. Setelah seluruh *chunked-file* dikirimkan ke bagian *server-side*, *client-side* akan mengirimkan identitas dari file yang asli ke halaman "/finish/". Untuk menyatakan bahwa file telah terkirim secara keseluruhan dan siap melakukan proses pengunggahan file kembali. Gambar 4.8 adalah kode program untuk mengirimkan identitas file ke *server-side*.

```
    var data = 'key=' + this.uploadData.key + '&name=' +
this.uploadData.file.name;
       xhr = new XMLHttpRequest();
       xhr.open("POST", parent.settings.scriptPath + '/finish/' +
this.uploadData.key, true);
       xhr.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
       xhr.onreadystatechange = function() {
              if(xhr.readyState == 4) {
                 var response = JSON.parse(xhr.response);
                 if(response.errorStatus !== 0 || xhr.status != 200) {
                    parent.printResponse(response.errorText, true);
                    return;
                 }

                 parent.resetKey();
                 parent.$(parent.settings.submitButton).value = 'Start
Upload';
                 parent.printResponse('File uploaded successfully.', false);
                 parent.success(response);
              }
           };
       xhr.send(data);
    };
```

Gambar 4. 8 Kode program pengiriman nama file.

Kode program pada gambar 4.8 berfungsi untuk mengirim data dari file ke halaman "/finish/" pada *localhost*, dimana variabel dari data akan berisi nama file asli sebelum dilakukan proses *chunking*. Kode program ini juga digunakan untuk memastikan bahwa seluruh *chunked-file* telah sampai pada *server-side*. Kemudian secara bersamaan juga dikirimkan ke *database* oleh *server-side*. Pada pemanggilan fungsi "parent.resetKey()", *client-side* akan mereset halaman pengunggahan file dan siap melakukan proses pengunggahan kembali. Proses tersebut ditunjukan dengan mengeluarkan *output* "*File Uploaded successfully*" sebagai indikasi yang menunjukan file telah dikirimkan.

4.1.3 *Metadata*

Metadata adalah data yang berisi informasi dari sebuah file. *Metadata* memiliki parameter yang akan dimasukan sebagai informasi dari sebuah file. Parameter ini dapat ditentukan berdasarkan kebutuhan pengembang dalam merancang sistem. *Metadata* dimasukan dalam proses pengiriman file, dan nantinya akan dibaca oleh *server-side*. Gambar 4.9 merupakan kode program yang menunjukan parameter yang digunakan dalam *metadata*.

```
Xhr.open(“POST”, parent.settings.scriptPath + ‘/upload/' +
                        parent.uploadData.key +
                        ‘/' + chunk +
                        “?filename=” + parent.uploadData.file.name +
                        “&filesize=” + parent.uploadData.file.size +
                        “&mimetype=” + parent.uploadData.file.type +
                        “&totalchunk=”  +  parent.uploadData.numberOfChunks,
true);
```

Gambar 4. 9 Kode program parameter *metadata*

Baris pertama dari kode program gambar 4.9 menunjukan *metadata* ini dikirim ke halaman “/*upload*/” pada localhost:8000. *Metadata* ini berisi parameter untuk menunjukan informasi dari file yang dikirimkan. Parameter tersebut adalah *filename, filesize*, *filetype,* dan *totalchunk*. Nantinya semua parameter ini akan disimpan dalam *database* sebagai informasi dari file asli. Karena parameter ini sangat diperlukan dalam pengelolaan *chunked-file* ketika telah disimpan dalam *database*.

4.1.5 Menghentikan Proses *Upload*

Ketika proses pengunggahan file dihentikan oleh pengguna, maka sistem akan menghentikan proses pengiriman file. *Client-side* menangani ini dengan mengirimkan *http-method* ke halaman “/abort” untuk menghentikan proses pengunggahan file. Gambar 4.10 adalah kode program untuk menghentikan proses pengunggahan.

```
this.abortFileUpload = function() {
       this.uploadData.aborted = true;
       var data = 'key=' + this.uploadData.key;
       xhr = new XMLHttpRequest();
       xhr.open("POST",    this.settings.scriptPath    +    '/abort/'    +
this.uploadData.key, true);
       xhr.setRequestHeader("Content-type",         "application/x-www-form-
urlencoded");
       xhr.onreadystatechange = function() {
              if(xhr.readyState == 4) {
                  var response = JSON.parse(xhr.response);

                  if(response.errorStatus !== 0 || xhr.status != 200) {
                     parent.printResponse(response.errorText, true);
                     return;
                  }
                  parent.printResponse('File upload was cancelled.', true);
              }
           };
       xhr.send(data);
   };
```

Gambar 4. 10 Kode program untuk menghentikan proses unggah

Kode program pada gambar 4.10 menunjukan fungsi untuk menghentikan proses pengiriman file. Pertama sistem memberikan nilai dari variabel menjadi "*true"* pada variabel digunakan untuk menghentikan proses. Kemudian mengirimkan nilai tersebut ke halaman "/abort/", agar sistem dapat menghentikan proses pengunggahan file. Setelah itu proses akan dihentikan oleh sistem dengan mengeluarkan output "*File upload was cancelled*".

4.1.6 Resume dan *Pause*

Penanganan untuk *resume* dan *pause* ketika proses pengunggahan berlangsung merupakan hal yang sangat penting. *Resume* dan *pause* selalu digunakan dalam setiap sistem yang memiliki fitur pengunggahan file. Oleh karena itu, sistem pada penelitian ini menggunakannya sebagai penanganan terhadap proses pengunggahan file, baik untuk menghentikan sementara atau melanjutkan. Penanganan semacam ini juga penting mengingat sistem ini menggunakan metode *chunking* yang membuat pengiriman dilakukan secara berkala. Gambar 4.11 adalah kode program untuk melakukan *resume* dan *pause*.

```
this.pauseUpload = function() {
       this.uploadData.paused = true;
       this.printResponse('', false);
       this.$(this.settings.submitButton).value = 'Resume';
   };
this.resumeUpload = function() {
       this.uploadData.paused = false;
       this.$(this.settings.submitButton).value = 'Pause';
       this.sendFile(this.uploadData.pauseChunk);
   };
```

Gambar 4. 11 Kode program untuk *resume* dan *pause.*

Kedua proses ini dilakukan dengan memberikan nilai dari variabel "*upload*Data.paused" yang berguna dalam menghentikan proses. Untuk proses *pause* memberikan nilai "*true*" dan untuk *resume* memberikan nilai "*false*". Dalam prakteknya, proses pause dengan nilai "*true*" akan menghentikan proses pengunggahan file. Namun *chunked-file* terakhir tetap disimpan di dalam "this.uploadData.pauseChunk" pada bagian pengiriman file. Sehingga proses pengunggahan file tetap dapat dilanjutkan kembali. Proses *resume* akan melanjutkan pengiriman file dengan membatalkan *method* "this.pauseUpload". Kemudian memasukan kembali proses pada potongan terakhir.

4.1.7 Konektivitas *Client-side* dan *Server-side*

Client-side dan *server-side* dihubungkan melalui "localhost:8000" dengan menggunakan *routing package* dari gorilla mux. Mux itu sendiri adalah *package* untuk bahasa pemrograman Go agar dapat membuat sebuah *http-router* pada sistem koneksinya. *Http-router* ini berjalan di localhost:8000, karena "8000" dianggap *port* yang tidak sedang digunakan oleh komputer maupun aplikasi lainnya. Sehingga dapat digunakan sebagai *http-router* sekaligus penghubung antara *client-side* dan *server-side*. *Client-side* dan *server-side* berkomunikasi didalamnya, *client-side* akan mengirimkan segala bentuk *http-method* ke localhost:8000. Sedangkan *server-side* juga akan mengambil atau mengirimkan segala bentuk *respond* maupun *request* melalui localhost:8000 ini. Disinilah konektivitas antara *client-side* dan *server-side* terjadi, dengan begitu proses pengiriman file dari pengguna dapat dikirim menuju *server-side*.

Gambar 4. 12 Konektivitas *client-side* dan *server-side*

Gambar 4.12 menunjukkan konektivitas antara *client-side* dan *server-side* yang terjadi melalui localhost:8000. Dalam prakteknya, segala bentuk pengiriman file dari *client-side* akan dikirimkan ke *server-side* melalui *http-router* ini. *Server-side* juga akan mengirimkan segala bentuk responnya ke *client-side* melalui localhost:8000 ini. Konektivitas ini dituliskan dalam *client-side* dengan mengarahkan semua bentuk pengiriman ke localhost:8000. Gambar 4.13 adalah kode program untuk mengkoneksikan *client-side* dengan localhost:8000.

```
function bigUpload () {
   this.settings = {
      'scriptPath': 'http://localhost:8000',
};
```

Gambar 4. 13 Koneksi ke localhost:8000

Baris 1 pada kode program adalah fungsi yang akan menyimpan pengaturan tentang semua bentuk pengiriman dari *client-side* ini. Kode program pada gambar 4.13 berguna untuk mengarahkan pengiriman ke localhost:8000. Dimana localhost:8000 merupakan penghubung

antara *client-side* dan *server-side*. Dengan adanya konektivitas antara kedua komponen penting pada sistem ini, maka telah sesuai dengan analisis yang dilakukan pada bab sebelumnya.

4.2 Server-side

4.2.1 Localhost

Seperti yang telah dijelaskan dalam sub-bab sebelumnya bahwa koneksi antara *server-side* dan *client-side* terjadi di localhost:8000. Agar localhost:8000 dapat menjadi *http-router*, maka harus menggunakan aplikasi *package routing* seperti Gorilla Mux. Pertama dilakukan instalasi Gorilla mux dengan perintah *command-line* seperti pada gambar 4.14.

```
Go get -u github.com/gorilla/mux
```

Gambar 4. 14 Instalasi Gorilla Mux

Setelah instalasi package Gorilla Mux selesai, maka siap digunakan dan diimplementasikan pada *server-side*. Untuk dapat mengakses *package* dari Gorilla Mux, *server-side* harus melakukan *import* kode program. *Import* ini dilakukan pada *server-side* agar dapat menggunakan localhost:8000 nantinya. Gambar 4.15 adalah kode program untuk meng*import package* dari Gorilla Mux sebelum digunakan dalam *server-side* nantinya.

```
package main
import (
    "bytes"
    "context"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "math/rand"
    "net/http"
    "os"
    "strconv"
    "time"

    "github.com/gorilla/mux"
```

Gambar 4. 15 *Import package* Gorilla Mux

Gambar 4.14 adalah proses *import package* dari Gorilla Mux yang telah diinstal sebelumnya. Proses *import* ini menungkinkan untuk mengakses isi dari *package* Gorilla Mux. Dengan begitu, *server-side* dapat diprogram untuk berkomunikasi melalui localhost:8000

dengan *client-side*. Konektivitas ini sangat penting mengingat sistem pada penelitian ini memproses pemindahan file dari pengguna ke *database*. Maka penting untuk memperhatikan konektivitas antara *client-side* dan *server-side*. Dengan melakukan *import* Gorilla Mux komunikasi 2 komponen ini dapat dimanipulasi sesuai kebutuhan sistem.

4.2.2 Konektivitas *Server-side* dan *Database*

Server-side dan *database* harus dapat berkomunikasi dengan baik, dikarenakan adanya pengiriman file dari *server-side* ke *database*. Secara teknis MongoDB dapat berkomunikasi dengan berbagai bahasa pemrograman pada *server-side*. Namun MongoDB membutuhkan *driver* sebagai penghubung antara *database* MongoDB dengan *server-side* yang menggunakan golang. MongoDB menyediakan *driver* tersebut dan memiliki nama "Mongo Go Driver". *Driver* tersebut merupakan *package* yang berisi kode program untuk mengkoneksikan MongoDB dengan Golang. Dimana *driver* tersebut dapat diinstal dengan perintah seperti gambar 4.16 di bawah ini.

```
Go get go.mongodb.org/mongo-driver
```

Gambar 4. 16 Instalasi MongoDB Go Driver

Setelah instalasi selesai maka *driver* tersebut dapat digunakan sesuai dengan kebutuhan pengembang. Sehingga dapat dilakukan koneksi antara *server-side* dengan *database* MongoDB dengan baik. Untuk penggunaan *driver* tersebut dapat dilakukan dengan *import* beberapa *package driver* pada *server-side*. *Import* ini bertujuan untuk memanggil beberapa *package* dari MongoDB Go Driver pada kode program. Setelah proses instalasi driver berhasil, secara otomatis *package* akan tertulis pada kode program *server-side*. Namun dapat juga dilakukan pemanggilan *package* secara manual dengan menuliskan beberapa kode program. Gambar 4.17 merupakan kode program untuk memanggil *package* dari Mongo Go Driver dan mengakses *database* MongoDB melalui *server-side*.

```
package main

import (
    "bytes"
    "context"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "math/rand"
    "net/http"
    "os"
    "strconv"
    "time"

    "github.com/gorilla/mux"
    "go.MongoDB.org/mongo-driver/bson"
    "go.MongoDB.org/mongo-driver/bson/primitive"
    "go.MongoDB.org/mongo-driver/mongo"
    "go.MongoDB.org/mongo-driver/mongo/gridfs"
    "go.MongoDB.org/mongo-driver/mongo/options"
)
```

Gambar 4. 17 *Import package* MongoDB Go Driver

Gambar 4. 17 menunjukan kode program untuk mengakses *package* dari *driver*. Setelah menuliskan kode program pada gambar, maka sekarang *server-side* dan MongoDB dapat dikoneksikan dengan baik. Konektivitas yang baik antara *server-side* dan *database* merupakan aspek penting dalam mencapai keberhasilan sebuah sistem. Sistem pada penelitian ini juga membutuhkan koneksi yang baik antara *server-side* dan *database*. Setelah melakukan im*port* package dari MongoDB Go Driver, secara teknis *server-side* dan *database* dapat terkoneksi. Untuk mengarahkan *server-side* ke *database* diperlukan URL (*Uniform Resource Locator*) yang mengarahkan koneksi ke MongoDB Atlas. MongoDB Atlas memiliki *url-link* untuk mengakses *cluster* dan *database* di dalamnya. Gambar 4.18 merupakan kode program untuk mengarahkan koneksi *server-side* menggunakan *url-link* ke *cluster* dan *database* MongoDB Atlas.

```
Var client *mongo.Client
var dbname = "bama"
var      dburl      =      "MongoDB://bama:1234567890@cluster1-shard-00-00-
xnpza.MongoDB.net:27017,cluster1-shard-00-01-
xnpza.MongoDB.net:27017,cluster1-shard-00-02-
xnpza.MongoDB.net:27017/test?ssl=true&replicaSet=Cluster1-shard-
0&47uthSource=admin&retryWrites=true&w=majority&ssl=true"
```

Gambar 4. 18 Koneksi ke MongoDB

Url-link ini didapatkan melalui pengaturan dari sisi *database*, dan merepresentasikan nama dari setiap *database* yang akan digunakan pada sebuah *cluster*. Dengan berisi juga *username*

dan *password* sebagai autentikasi pemilik dari *cluster* tersebut. Angka 27017 merupakan *default-port* dari *database MongoDB* yang secara *default* berjalan pada *port* tersebut. Setelah menuliskan kode program pada gambar 4.18, maka segala bentuk perintah pengiriman akan ditujukan pada url-link tersebut. *Server-side* akan mengirimkan file dari pengguna menuju 3 jenis *database* di dalam *cluster* pada MongoDB yaitu 1 *primary database* dan 2 *secondary database*. Hal ini dapat dilihat dari *url-link* yang telah dituliskan pada kode program, *url-link* ini menuju pada 3 *database* sekaligus. Dengan begitu perintah untuk mengunggah file siap dijalankan dan akan disimpan dengan baik di *database* MongoDB.

4.2.3 Id-file Sebagai *Filename*

Dalam setiap media penyimpanan dalam sebuah sistem, diperlukan id untuk identitas bagi setiap file yang akan disimpan di dalamnya. Id ini berfungsi sebagai identitas setiap file yang ada pada sebuah *database*. Id itu sendiri juga memiliki fungsi sebagai pembeda antara setiap file di *database*. Oleh karena itu id yang disematkan pada sebuah file haruslah unik atau tidak memiliki kesamaan dengan file lain. Apalagi dengan model *unstructured database* yang fleksibel dan tidak menggunakan relasi antara tabel di dalamnya. Pada penelitian ini id file dibuat untuk memberi identitas untuk antara *chunked-file* satu dengan lainnya. Pemberian id file dilakukan pada *server-side*, agar setiap file yang akan disimpan mudah dikelola kembali. Gambar 4.19 merupakan parameter "id" yang disematkan ke dalam file yang akan disimpan.

```
uploadStream, err := bucket.OpenUploadStream(
       status.Key + "." + chunkNumber,
   )
```

Gambar 4. 19 Parameter untuk *filename.*

Setiap file diberi id yang unik yaitu "*Key*" dan "chunkNumber", Karena parameter tersebut tidak akan sama antara file satu dengan file lainnya. "*Key*" adalah variabel yang dideklarasikan pada *server-side* dan didapat dari proses "generate*Key*". Hasil dari *generate* ini memiliki kode unik untuk setiap kelompok *chunking* file satu dengan lainnya. Namun memiliki kesamaan antara *chunked-file* dalam satu kelompok file. Sehingga diperlukan parameter tambahan yaitu "chunkNumber" yang merupakan nomor untuk setiap *chunked-file*. Dengan begitu setiap *chunked-file* dalam satu kelompok memiliki identitas yang unik, juga antara kelompok *chunked-file* satu dengan kelompok *chunked-file* lainnya.

4.2.4 *Upload* File

Sebelum melakukan pengunggahan file *server-side* terlebih akan memberikan "*key*" untuk setiap *chunked-file*. *Key* ini dapat juga digunakan untuk membedakan file yang akan disimpan dalam *database* nantinya. Sebagai contoh 2 file yang sama diunggah ke sistem, maka kedua file tersebut akan memiliki *key* untuk nama file yang berbeda. *Key* ini di dapatkan dengan melakukan *generate key*. Gambar 4.20 dibawah ini adalah kode program untuk melakukan *generate key*.

```
const charset = "abcdefghijklmnopqrstuvwxyz" +
"ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

func generateKey(length int) string {
    return stringWithCharset(length, charset)
}
```

Gambar 4. 20 Proses *generate key*

Baris ke-1 dari kode adalah memasukan karakter yang akan dilakukan pengacakan dalam fungsi *generateKey*. Fungsi ini akan menghasilkan *key* yang unik untuk setiap file. Oleh karena itu *key* ini akan digunakan sebagai nama file ketika proses pengiriman. Setelah itu proses dari *server-side* adalah menerima *chunked-file* dari *client-side* melalui localhost:8000. Untuk dapat dikirimkan menuju *database*, *server-side* perlu menerima *request* dari *client-side* secara spesifik. Karena *client-side* mengirimkan setiap bentuk *request*nya ke halaman-halaman yang ada pada localhost:8000. Gambar 4.21 dibawah ini adalah kode program untuk mengakses tiap halaman yang akan berisi file dari *client-side*.

```
    router := mux.NewRouter()
    router.HandleFunc("/upload/{key}/{chunk_number}",
uploadFile).Methods("POST")
    router.HandleFunc("/abort/{key}", abortUpload).Methods("POST")
    router.HandleFunc("/finish/{key}", finishUpload).Methods("POST")
    router.HandleFunc("/download/{key}", downloadFile).Methods("GET")

    http.ListenAndServe(":8000", router)
}
```

Gambar 4. 21 Kode program untuk *routing*

Baris ke-1 berguna untuk mengkoneksikan *server-side* dengan mux, dimana mux ini berfungsi sebagai *http-router*. Setelah itu variabel *router* ini berisi *http-method* untuk mengakses tiap halaman pada localhost:8000. Dimana setiap halaman ini akan berisi *request*

dari *client-side*. Seluruh *chunked-file* yang akan dimasukan dalam *database* juga berada didalamnya.

```
Func uploadFile(w http.ResponseWriter, r *http.Request) {
w.Header().Set("Content-Type", "application/json")
w.Header().Set("Access-Control-Allow-Origin", "*")
var status Status
status.ErrorStatus = int(900)

params := mux.Vars®
key := params["key"]
chunkNumber := params["chunk_number"]

filename := r.URL.Query().Get("filename")
fileSize, _ := strconv.Atoi(r.URL.Query().Get("filesize"))
totalChunk, _ := strconv.Atoi(r.URL.Query().Get("totalchunk"))
mimeType := r.URL.Query().Get("mimetype")

if key == "0" {
    key = fmt.Sprintf("%s%s", generateKey(10), 50ilename)
    collection := client.Database(dbname).Collection("fileMetadata")
    file := FileMetadata{key, 50ilename, fileSize, mimeType, totalChunk}
    result, err := collection.InsertOne(context.TODO(), file)
    if err != nil {
        fmt.Println(err)
        os.Exit(1)
    }
    log.Println(result)
}

status.Key = key
log.Printf("key id: %s\n", key)
log.Printf("chunk number: %s\n", chunkNumber)

byts, err := ioutil.ReadAll(r.Body)
if err != nil {
    log.Fatal(err)
    json.NewEncoder(w).Encode(status)
}

if byts == nil {
    log.Fatal("Bytes data is null")
    json.NewEncoder(w).Encode(status)
}

bucket, err := gridfs.NewBucket(
    client.Database(dbname),
)

if err != nil {
    log.Fatal(err)
    os.Exit(1)
}
uploadStream, err := bucket.OpenUploadStream(
    status.Key + "." + chunkNumber,
)
if err != nil {
    fmt.Println(err)
```

```
        os.Exit(1)
    }
    defer uploadStream.Close()

    dataSize, err := uploadStream.Write(byts)
    if err != nil {
        log.Fatal(err)
        os.Exit(1)
    }
    log.Printf("Write file to DB was successful. File size: %d M\n", dataSize)

    status.ErrorStatus = int(0)
    json.NewEncoder(w).Encode(status)
}
```

Gambar 4. 22 Kode program *upload file*.

Proses pengunggahan dari *client-side* ini dimulai dengan mengakses *metadata* dari *chunked-file*. Kode program pada gambar 4.22 menunjukan *metadata* ini diakses oleh *server-side*. Dimana *metadata* ini telah diproses dan didapatkan dari bagian *client-side*. Setelah itu *server-side* akan mencari file dengan *key* 0 yang berarti *metadata*, dan akan dikirimkan ke *database* pada *collection* "file*Metadata*". Setelah itu seluruh *chunked-file* akan dikirim dan disimpan dalam *database*.

4.2.5 *Download* File

Proses pengunduhan file dilakukan oleh *server-side* dengan mengirimkan file langsung ke pengguna melalui aplikasi *web-browser*. Gambar 4.23 adalah proses pengunduhan dari bagian *server-side*.

```
Func downloadFile(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    key := params["key"]

    var file FileMetadata
    collection := client.Database(dbname).Collection("fileMetadata")
    log.Println(key)

    filter := bson.D{primitive.E{"key", key}}
    err := collection.FindOne(context.TODO(), filter).Decode(&file)

    if err != nil {
        http.NotFound(w, r)
        return
    }

    log.Println(file.Key)
    log.Println(file.Name)
    log.Println(file.Size)
    log.Println(file.TotalChunk)
    log.Println(file.Type)
```

```
    bucket, err := gridfs.NewBucket(
        client.Database(dbname),
    )

    if err != nil {
        log.Fatal(err)
    }

    flusher, ok := w.(http.Flusher)
    if !ok {
        panic("expected http.ResponseWriter to be an http.Flusher")
    }
    fileSize := fmt.Sprintf("%d", file.Size)
    w.Header().Set("X-Content-Type-Options", "nosniff")
    w.Header().Set("Content-Type", file.Type)
    w.Header().Set("Content-Length", fileSize)
    w.Header().Set("Content-Disposition", "attachment; filename="+file.Name)
    for i := 0; i <= file.TotalChunk; i++ {
        // fmt.Fprintf(w, "Chunk #%d\n", i)
        var buf bytes.Buffer
        datakey := fmt.Sprintf("%s.%d", file.Key, i)
        dStream, err := bucket.DownloadToStreamByName(datakey, &buf)
        if err != nil {
            return
        }
        log.Printf("File size to download: %v\n", dStream)
        buf.WriteTo(w)
        flusher.Flush() // Trigger "chunked" encoding and send a chunk…
        time.Sleep(500 * time.Millisecond)
    }

}
```

Gambar 4. 23 Kode program untuk *download file*.

Proses *download file* diawali dengan mengakses *database* pada *collection* "file*Metadata*". Kemudian mengeluarkan *output* berdasarkan *key* dari file tersebut. Proses pengunduhan dilanjutkan dengan mengeluarkan *output* yang berisi *metadata* dari file yang akan diunduh. Lalu proses pengunduhan berjalan dari *chunked-file* pertama hingga *chunked-file* terakhir.

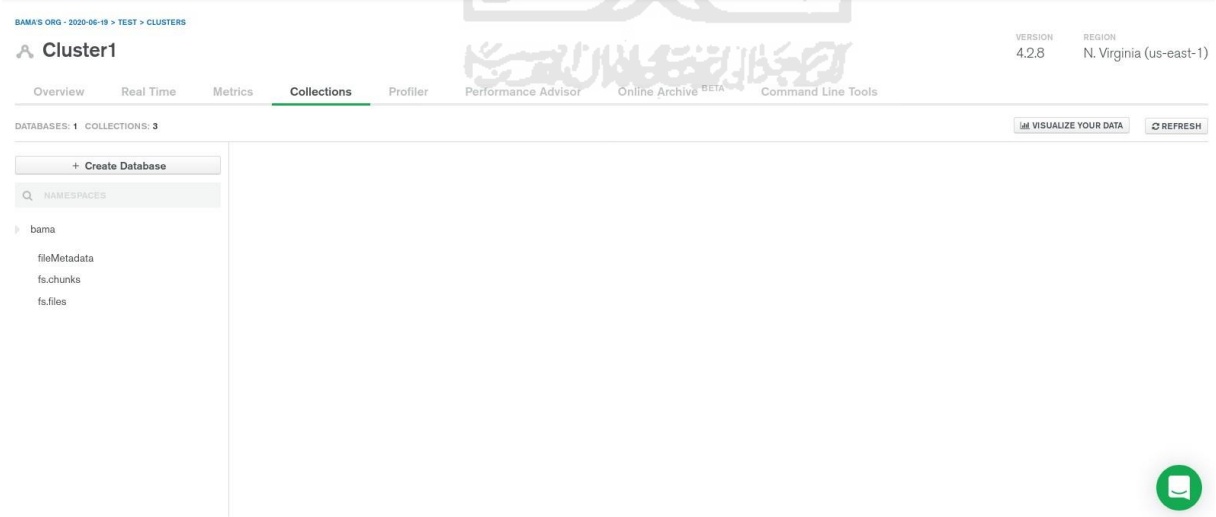
4.3 Database

Pada bab 3 sebelumnya telah dilakukan pembahasan tentang *unstructured database*. Sistem pada penelitian ini menggunakan salah satu produk yang menggunakan model *unstructured database* yaitu MongoDB. MongoDB adalah salah satu produk *database* berjenis *unstructured database* yang bersifat NoSQL. Produk ini merupakan *database* yang paling populer di industri teknologi informasi saat ini. MongoDB dapat diakses pada *command-line* atau dalam versi GUI (*Graphical User Interface*). MongoDB mengeluarkan produk veri GUI dengan nama MongoDB Atlas, yang sebenarnya sama dengan versi *command-line*. Namun

versi GUI ini memudahkan pengembang dalam mengelola *database* di dalam sebuah sistem. Tidak seperti *database* konvensional lainnya, MongoDB Atlas membutuhkan beberapa pengaturan agar dapat berintegrasi maksimal dengan komponen lainnya pada sistem.

4.3.1 Pengaturan *Database*

Terdapat beberapa cara yang bisa dilakukan untuk menggunakan *database* MongoDB, seperti meng*install* pada komputer dan menjalankannya pada *command-line*. Cara lain yaitu dengan menggunakan *mode Graphical User Interfaces* (GUI) dari MongoDB, yaitu MongoDB Atlas. MongoDB Atlas adalah produk GUI dari MongoDB, agar pengguna dapat mengakses *database*nya dengan mode GUI. Sebelumnya untuk mengakses MongoDB Atlas diperlukan registrasi. Setelah registrasi berhasil maka dapat dilakukan pengaturan didalamnya. Sama seperti yang telah dianalisa pada bab sebelumnya, pada bagian *database* ini dilakukan replikasi pada *primary database*. MongoDB atlas memiliki fitur untuk dimungkinkan adanya 3 *database* yaitu 1 *primary database* dan 2 *secondary database*. Tidak seperti pada *database* konvensional, pengaturan *database* MongoDB tidak diperlukan ERD atau skema relasi antara satu tabel dengan tabel lainnya. Sehingga pengaturan pada *database* ini terbilang cukup mudah dan tidak membutuhkan waktu yang lama. Gambar 4.24 dibawah ini merupakan tampilan dari *database* MongoDB yang digunakan dalam sistem pada penelitian ini.



Gambar 4. 24 *Cluster* MongoDB Atlas.

Pada bagian *database* ini memiliki nama “bama” yang merupakan representasi nama dari pemilik *database*. Pada pojok kanan atas terdapat tulisan “N. Virginia” yang merupakan lokasi dari *database* ini. *Cluster* 1 yang tertulis pada gambar adalah kelompok *database*, dimana dalam 1 *cluster* dapat dibuat banyak *database*. Namun penelitian ini menggunakan 1 *cluster* dan 1 *database* saja di dalamnya. Karena sistem pada penelitian ini hanya memiliki 2 proses yaitu unggah dan unduh file. *Database* MongoDB tidak menggunakan tabel, namun menggunakan *collections* sebagai tempat untuk menyimpan dokumen nantinya. Dalam pengaturan ini dirancang 3 *collection* yaitu file*Metadata*, fs.chunks, dan fs.file, masing-masing *collection* menyimpan dokumen sesuai dengan arahan dari *sever-side*. *Collection* “file*Metadata*” dirancang untuk menyimpan *metadata/identifier-key* pada setiap file yang akan disimpan nantinya. *Collection* fs.chunks adalah *collection* yang berisi *default chunking* dari sistem MongoDB. Dimana *chunked-file* dari file pengguna yang akan disimpan dilakukan *chunking* kembali oleh sistem dan disimpan dalam *collection* “fs.chunks”. Untuk *collection* “fs.file” berisi *chunked-file* dari file yang di unggah oleh pengguna. *Collection* “fs.file ”ini merupakan *collection* yang terpenting dari *database* ini, karena berisi file pengguna yang telah dipecah menjadi beberapa bagian. Setelah mengatur *database* beserta *collection* didalamnya, selanjutnya dilakukan pengaturan mengenai replikasi *database*.

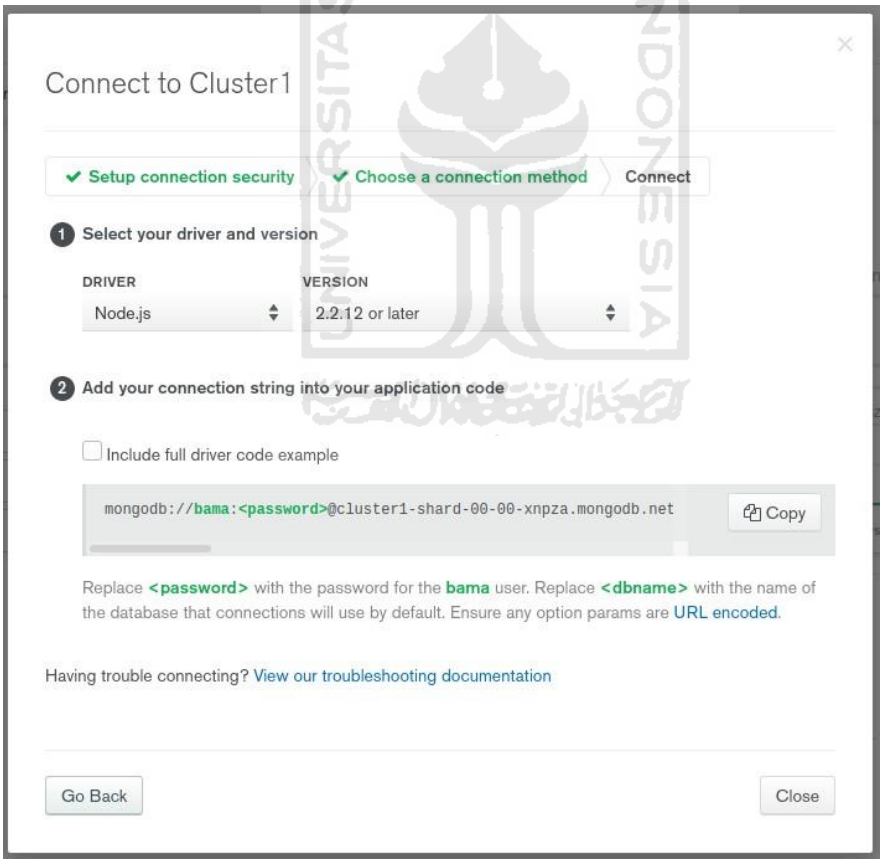
Gambar 4. 25 *Primary* dan *secondary database*.

Setiap model *unstructured database* biasanya memiliki fitur replikasi didalamnya, sama halnya dengan MongoDB. Replikasi pada MongoDB merupakan fitur yang dapat diatur sesuai dengan kebutuhan sistem. Adanya replikasi pada *database* dapat meningkatkan aspek *high-availability* atau tingkat ketersediaan file pada *database*. Karena file disimpan dalam 2 *database* cadangan dengan isi yang sama dengan *database* utama. Dengan fitur replikasi yang

telah diatur pada *database* ini, maka pengaturan *database* ini telah sesuai berdasarkan analisa dan perancangan pada bab sebelumnya.

4.3.2 Url-link Database

MongoDB Atlas memiliki URL (Uniform Resource Locator) untuk mengarahkan *server-side* menuju *database* di dalamnya. *Url-link* ini bisa didapatkan melalui pengaturan pada *cluster* dalam MongoDB Atlas. *Url-link* ini dapat diatur sesuai dengan kebutuhan pengembang, apakah ingin mengarahkan *server-side* menuju satu *database* saja atau 3 *database* sekaligus. Untuk sistem pada penelitian ini dilakukan pengaturan untuk *url-link* yang dapat mengkoneksikan dengan 3 *database* di dalamnya yaitu 1 *primary database* dan 2 *secondary database*. Gambar 4.26 adalah pengaturan untuk mendapatkan url-link dari *cluster* MongoDB Atlas.



Gambar 4. 26 Pengaturan *url-link*

Url-link dari *cluster* MongoDB Atlas ini akan digunakan oleh *server-side* dalam mengarahkan koneksi ke *database*. Sebelum mengimplementasikan *url-link* ke dalam *server-*

side perlu beberapa modifikasi yaitu pada autentikasi. Diperlukan untuk mengganti *username* dan *password* sesuai dengan yang nama dan kata sandi yang telah dibuat sebelumnya. *Url-link* ini akan mengarahkan *server-side* ke dalam 3 *database* di dalam *cluster* pada MongoDB Atlas. Hal tersebut dikarenakan kebutuhan sistem yang mengharuskan *server-side* mengakses 3 *database* sekaligus. Setelah semua komponen diimplementasikan sesuai analisis maka sistem secara keseluruhan siap untuk dilakukan pengujian.

4.4 Pengujian

Pengujian dilakukan untuk mengetahui tingkat keberhasilan sistem dalam menghadapi masalah yang diangkat. Sistem ini dirancang untuk dapat digunakan dalam keadaan internet yang *unstable* dan *unreliable*. Sehingga diperlukan skenario pengujian yang merepresentasikan masalah yang dihadapi. Tahap pengujian ini dilakukan dengan harapan bahwa sistem dapat berjalan dengan baik dan sesuai tujuan. Namun tidak menutup kemungkinan sistem ini tidak dapat bekerja sesuai dengan harapan sebelumnya. Upaya perancangan telah dilakukan semaksimal mungkin, apabila sistem tidak bekerja sesuai harapan. Maka tahapan ini menjadi bukti sekaligus pengetahuan baru tentang metode *chunking* dalam sistem pengunggahan file. Karena tujuan utama dari penelitian ini adalah mengkaji metode *chunking* dalam menghadapi masalah pengunggahan file. Apabila metode *chunking* memiliki banyak kelemahan, itu merupakan pengetahuan baru yang dapat dikembangkan lebih lanjut kedepannya. Upaya pembuktian sistem ini dilakukan dalam beberapa tahap dan skenario. Berikut adalah skenario pengujian yang akan dilakukan.

a. Mengunggah file dengan ukuran 10 MB dan 100 MB.

Tujuan dari skenario ini adalah untuk menguji sistem dalam hal pengunggahan file secara normal. Skenario pengujian akan mendapatkan informasi tentang proses *chunking* pada ukuran file yang berbeda. Sistem diuji dalam konsistensi untuk melakukan pengiriman file dari ukuran file yang kecil hingga ukuran file yang tergolong besar. Hal ini dikarenakan pengguna memiliki ukuran file yang bervariasi. Sehingga sistem nantinya harus melayani file dari pengguna dengan kinerja yang sama.

b. Ketersediaan file

Skenario ini merujuk pada aspek *high-availability* sistem, dimana sistem akan diuji untuk melakukan layanan pengunduhan file. Skenario ini juga dapat menjadi pengetahuan tentang

metode *chunking*. Dimana sistem dapat memberikan file semula dari pengguna yang sebelumnya telah diunggah menjadi *chunked-file*. Karena file asli dari pengguna tidak disimpan dalam kondisi yang utuh, melainkan terpotong-potong. Pengujian ini dilakukan dengan mengunduh file pengguna melalui *download-link* pada antarmuka. *Download-link* ini menuju pada pengunduhan file yang sebelumnya telah diunggah pengguna.

c. Kehandalan sistem

Skenario pengujian ini representasi internet pengguna yang terputus ketika proses pengunggahan file dilakukan. Skenario ini menguji seberapa handal sistem dengan metode *chunking* dalam menghadapi masalah internet yang *unreliable*. Sistem akan diuji untuk dapat membuktikan adanya *resumable-process* dalam hal pengiriman file. Karena sistem ditujukan untuk menyelesaikan masalah pengunggahan file dalam kondisi internet yang tidak memadai. Skenario ini dilakukan secara acak, yaitu memutus koneksi pada persentase angka yang acak.

d. Pengecekan *hash* menggunakan MD5

Pengujian ini dilakukan dengan melakukan pengecekan *hash* terhadap file yang telah diunduh. *Hash* menggunakan MD5 pada file tersebut dibandingkan dengan file semula yang telah diunggah sebelumnya. Kemudian dilakukan pengamatan apakan nilai dari MD5 kedua file serupa dan identik. Kedua file akan dinyatakan identic dan tidak mengalami perubahan sedikitpun apabila memiliki nilai *hash* yang sama.

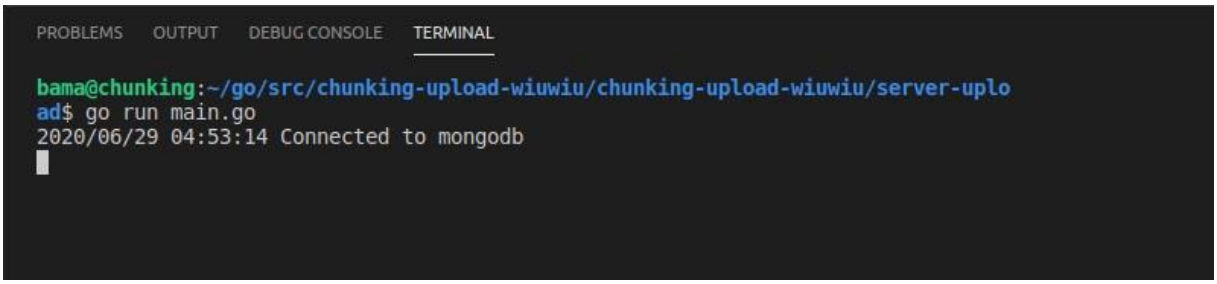
Sebelum melakukan pengujian, setiap komponen lebih dahulu disiapkan untuk dijalankan. Pada bagian *client-side* dan *database* tidak memerlukan aktivasi, karena dapat dijalankan hanya dengan membukanya. Untuk bagian *server-side* yang menjadi penghubung antara *client-side* dan *database*. Diperlukan aktivasi dengan menjalankan perintah pada *command-line*. Gambar 4.27 merupakan perintah untuk menjalankan *server-side*.

```
go run main.go
```

Gambar 4. 27 Menjalankan *server-side*.

Perintah ini dijalankan melalui terminal pada direktori dimana file “main.go” ini disimpan. Dengan menjalankan perintah ini, maka *server-side* telah dapat digunakan dalam sistem.

Pembuktian ini ditunjukan dengan *server-side* yang mengeluarkan output bahwa telah terkoneksi dengan *database*. Seperti yang ditunjuk pada gambar berikut.



Gambar 4. 28 *Server-side* telah terkoneksi.

Setelah *server-side* terkoneksi, maka pengujian dapat dilakukan dengan membuka *client-side*. *Client-side* ini berupa file html sebagai antarmuka sistem, namun file *html* ini telah berintegrasi dengan file *javascript* "bigUpload.js". Untuk bagian *database*, cukup membukanya dengan *login* pada MongoDB Atlas. Persiapan untuk pengujian telah selesai, dan siap dijalankan sesuai dengan skenario. Berikut adalah rangkaian pengujian yang dilakukan.

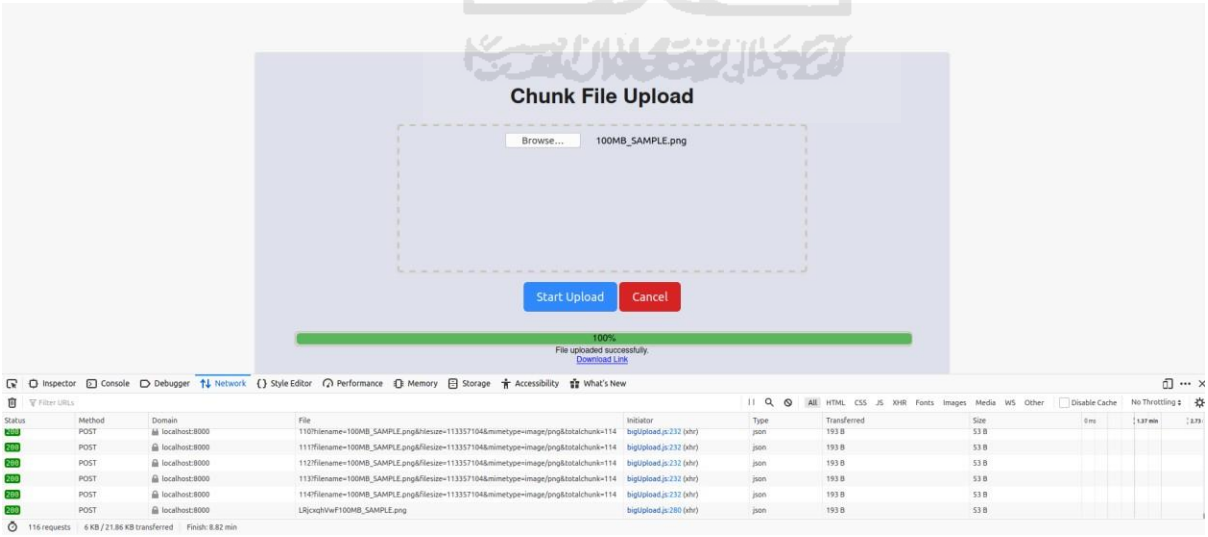
4.4.1 Mengunggah file dengan ukuran 10 MB dan 100 MB

Pada skenario ini dipilih file dengan ukuran 10 MB dan 100 MB, untuk file 10 MB digunakan image yang berekstensi jpg atau *Joint Photographic Group*. Untuk image berukuran 100 MB digunakan file berekstensi "png" atau *Portable Network Graphics*. Kedua file tersebut digunakan untuk mewakili ukuran terkecil dan terbesar untuk file yang sering diunggah pengguna. Artinya apabila dengan ukuran 100 MB, sistem dapat melakukan pengunggahan file dengan baik. Maka file dibawah 100 MB pun akan ditangani dengan baik juga. Namun, dalam prakteknya pengguna jarang mengunggah file dengan ukuran yang bulat puluhan seperti 10 dan 100. Maka dipilih ukuran file yang mendekati angka tersebut. Sebelum memulai skenario ini, file terlebih dahulu disiapkan. Termasuk dalam penamaan file untuk lebih mudah dalam mengidentifikasi. Karena file tetap akan disimpan dalam potongan-potongan dengan ukuran 1 MB. Gambar 4.29 adalah file yang digunakan dalam pengujian pada tahap ini.

Gambar 4. 29 *Sample* file untuk pengujian.

Kedua file diatas memiliki ukuran yang hampir sesuai dengan ukuran file yang digunakan dalam skenario pengujian. Kedua file diberi nama sesuai dengan ukuran masing-masing dan ditambah dengan kata "*sample*", untuk menunjukan file tersebut adalah file pengujian. Berikut adalah rangkaian pengujian pada setiap ukuran file.

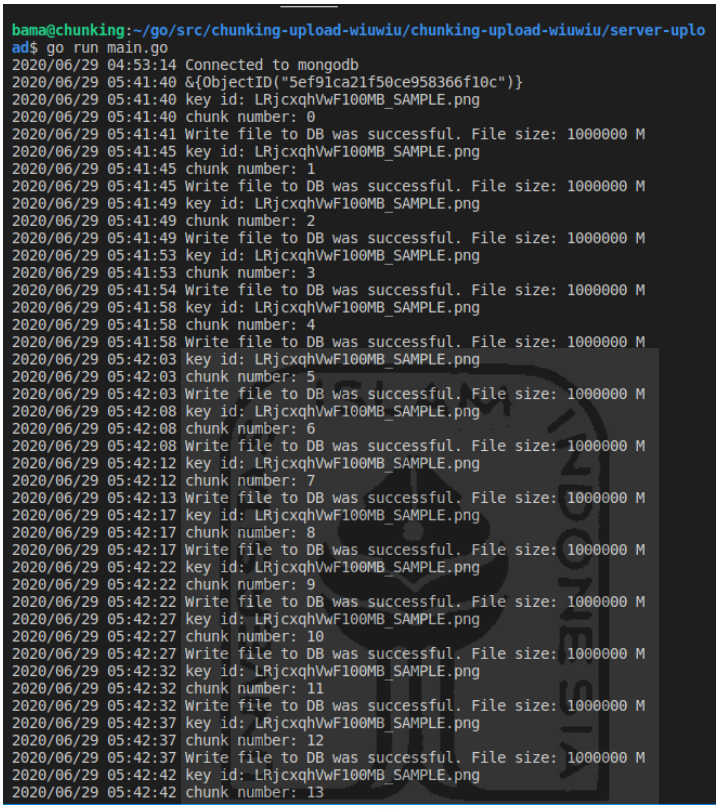
Pertama, file "100MB_SAMPLE" dengan ukuran 100 MB dimasukkan ke dalam sistem untuk diunggah. Kemudian dilakukan observasi untuk melihat proses yang bekerja pada setiap bagian sistem. Selain itu sistem diuji dengan melihat apakah file tersebut dapat disimpan dengan baik di *database*. Skenario ini termasuk juga menguji setiap bagian dari sistem, untuk menentukan bagian mana yang perlu ditingkatkan. Gambar 4.30 adalah proses pengujian dengan file berukuran 100 MB.



Gambar 4. 30 Proses pengiriman dari *client-side*.

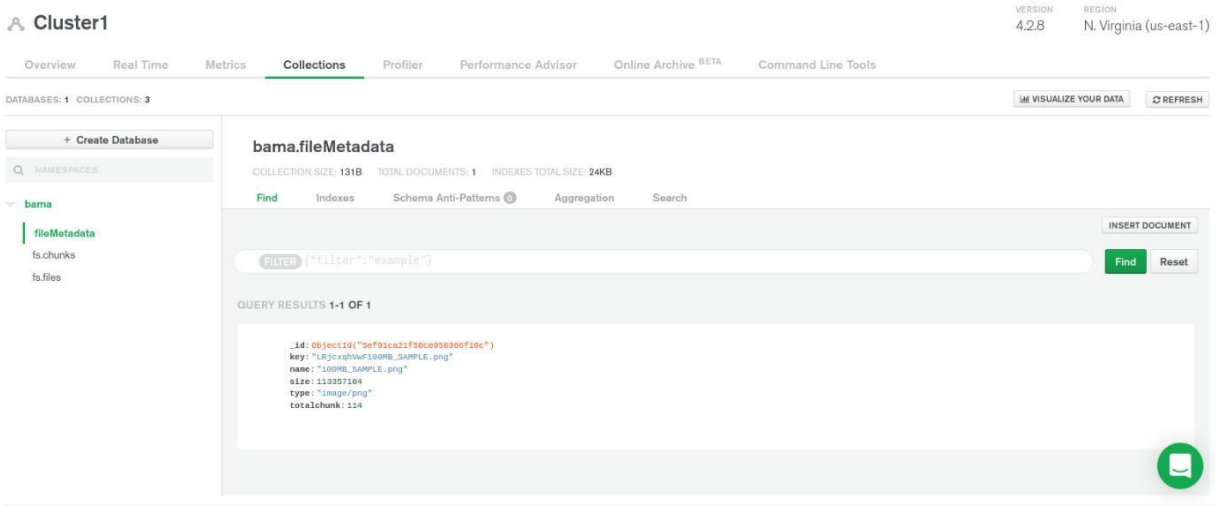
Proses pengunggahan menunjukan bahwa sistem memotong file "100MB_SAMPLE" menjadi 114 potongan. Dimana telah sesuai dengan apa yang dirancang dalam *client-side*

bagian *chunking*. Terlihat bahwa *client-side* telah mengirimkan potongan-potongan file ke localhost:8000 dengan *http-method* "POST". *Server-side* yang telah terkoneksi dengan localhost:8000 menerima file tersebut untuk kemudian mengirimkan ke *database*. Gambar 4.31 adalah proses pengiriman file dari *server-side* ke *database*.



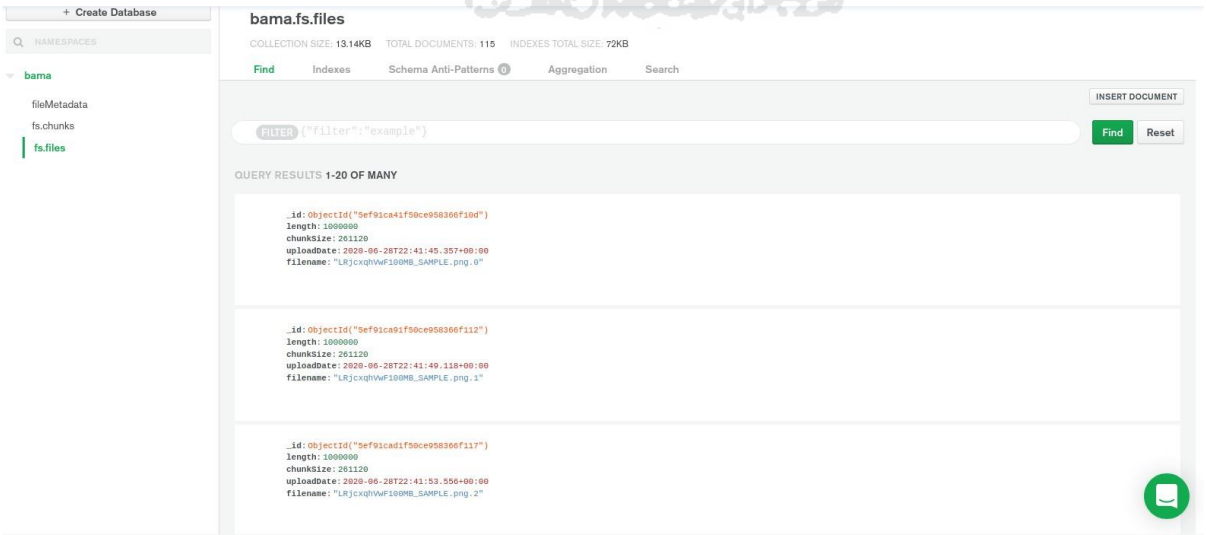
Gambar 4. 31 Proses yang terjadi di *server-side*.

Proses pengiriman setiap *chunked-file* dari *server-side* ke *database*, ditunjukan dengan output "Write file to DB was successful". Hasil ini menunjukan file yang telah dipotong oleh *client-side* telah berhasil dimasukan ke *database* melalui *server-side*. Setelah proses pengunggahan selesai, dilakukan pengecekan pada bagian *database*. Untuk menunjukan apakah file tersebut dapat disimpan sesuai dengan rancangan sistem. Pada *database* ini juga dilakukan pengamatan terkait apakah *metadata* dan *id-file* telah berhasil disematkan dalam *chunked-file*. Gambar 4.32 dibawah ini adalah hasil yang ditunjukan pada bagian *database*.



Gambar 4. 32 *Collection* "fileMetadata".

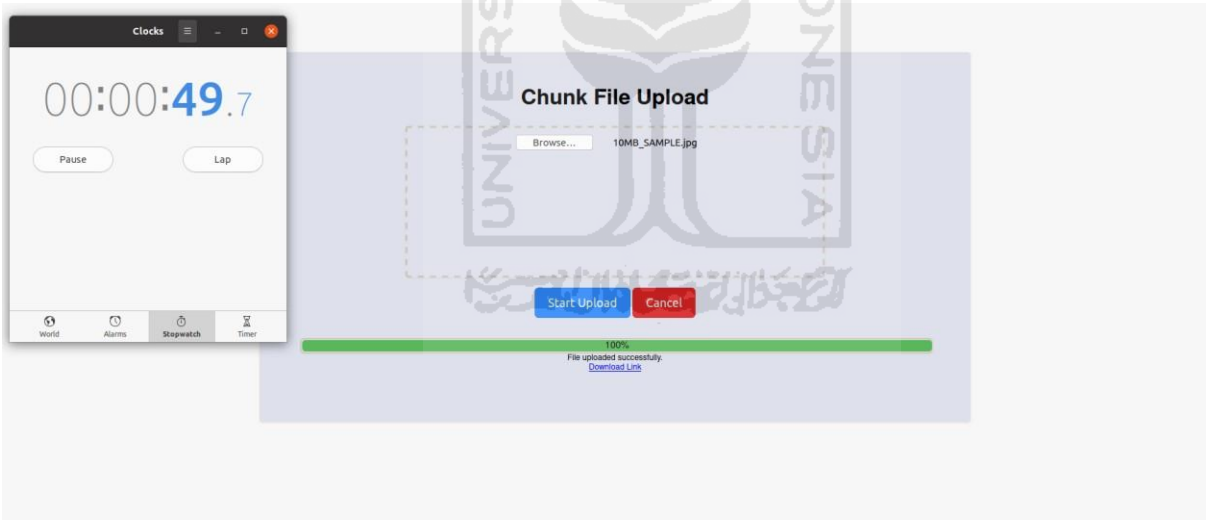
Database dengan nama "bama" menunjukan telah menyimpan file dalam setiap *collection*-nya. Pada *collection* "fileMetadata" terlihat bahwa *metadata* untuk file "100MB_SAMPLE" telah teridentifikasi dan disimpan dengan baik. Dengan parameter *key, name, size, type*, dan *totalChunk*. Semua parameter telah sesuai dengan file yang diunggah dalam tahap pengujian ini. *Metadata* ini merujuk pada kumpulan *chunked-file* untuk file "100MB_SAMPLE" yang disimpan dalam *collection* yang berbeda. Untuk melihat tiap potongan file yang tersimpan dapat dilihat dalam *collection* "fs.file". Gambar 4.33 adalah isi dari *collection* "fs.file" setelah pengujian dilakukan.



Gambar 4. 33 *Collection* "*fs.file*" setelah pengujian.

Collection “fs.file” telah terisi *chunked-file* dari file “100MB_SAMPLE”, dengan ukuran tiap potongan adalah 1 MB. Jumlah *chunked-file* untuk file “100MB_SAMPLE” adalah 114 dan setiap *chunked-file* memiliki *id-file* yang unik. Ditunjukan dari label “Filename” yang merujuk pada format “key.chunknumber” yang disematkan dalam file. Dimana parameter *key* didapat dengan *generate* abjad yang telah dijelaskan dalam bab sebelumnya. Untuk parameter “chunknumber” didapat dari urutan *chunked-file* yang terunggah. Sehingga setiap *chunked-file* dapat dikelola tanpa adanya kesalahan pemanggilan *chunked-file* pada *database*.

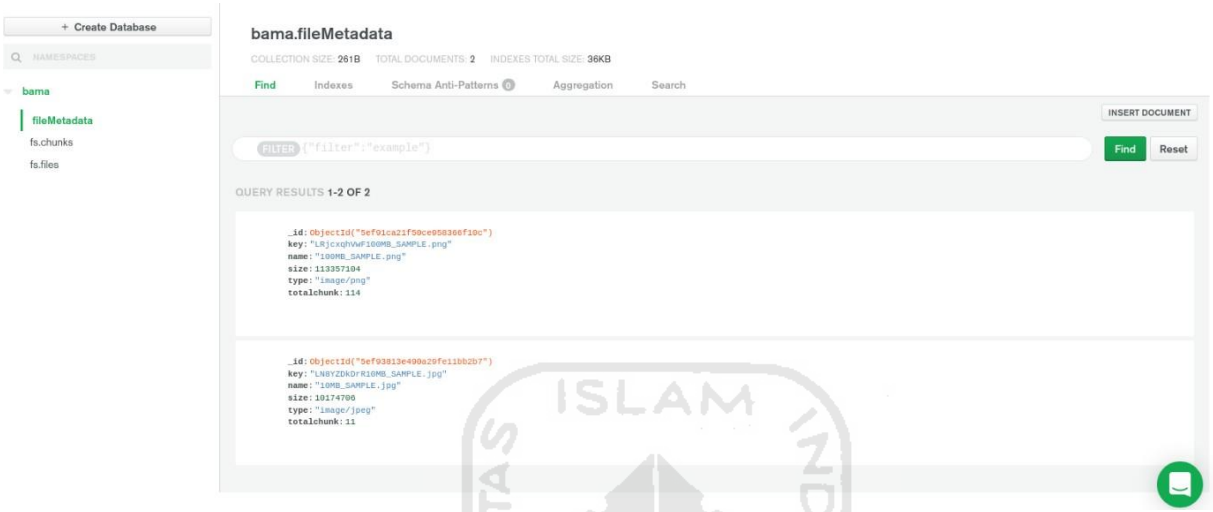
Setelah pengujian terhadap file “100MB_SAMPLE” dengan ekstensi png selesai, Pengujian dilanjutkan dengan pengunggahan file “10 MB_SAMPLE” dengan ekstensi jpg. Dengan urutan langkah pengujian yang sama dengan pengujian pada file “100MB_SAMPLE”. Namun pengujian pada file 10 MB ini ditambah dengan penggunaan *stopwatch* untuk melihat waktu yang dibutuhkan dalam pengunggahan file. Gambar 4.34 di bawah ini adalah hasil dari pengujian pada file “10 MB_SAMPLE”.



Gambar 4. 34 Waktu yang dibutuhkan dalam pengunggahan file.

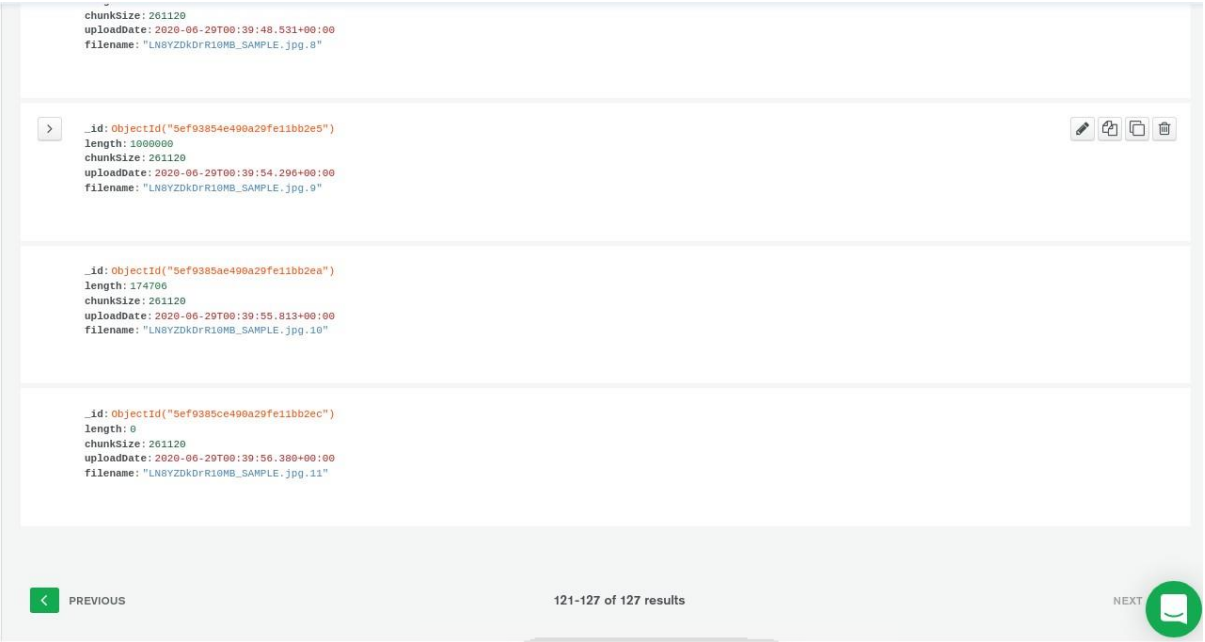
Kali ini pengujian dilakukan dengan menggunakan *stopwatch*, untuk mengetahui waktu yang diperlukan sistem dalam mengunggah file. Pengujian dilakukan untuk menguji sistem yang harus mengirimkan file dengan lokasi *database* yang jauh yaitu di North Virginia, US. Karena jarak sangat berpengaruh dengan kecepatan pengunggahan file. Kemudian dilakukan observasi tentang apakah sistem dapat mengirimkan dalam waktu yang cepat dengan lokasi *database* yang jauh dari *client-side* dan *server-side*. Pengujian dilakukan dengan rangkaian

yang sama dengan pengujian file “100MB_SAMPLE”. Dengan ukuran file yang lebih kecil. apakah sistem dapat menyimpan file ke dalam *database* dengan sama baiknya. Gambar 4.32 adalah hasil yang ditunjukan *database* setelah pengujian menggunakan file “10MB_SAMPLE”.



Gambar 4. 35 *Collection* “*fileMetadata*” pengujian ke-2.

Database menyimpan *metadata* dari file “10MB_SAMPLE” sama seperti ketika menyimpan file “100MB_SAMPLE”. Dalam *metadata* menunjukan bahwa file dipotong menjadi 11 bagian dan diidentifikasi sebagai image dengan *key* seperti pada gambar 4.32. Apabila melihat pada *collection* fileMetadata, sistem melakukan hal yang sama terhadap file “10MB_SAMPLE”. Mulai dari proses pengiriman hingga penyimpanan *database*, sistem memproses dengan cara yang sama. Untuk melihat setiap *chunked-file* yang disimpan dapat dilakukan dengan membuka *collection* “fs.file”. Gambar 4.33 adalah hasil yang ditunjukan *database* setelah pengujian dengan file “10MB_SAMPLE” dilakukan.



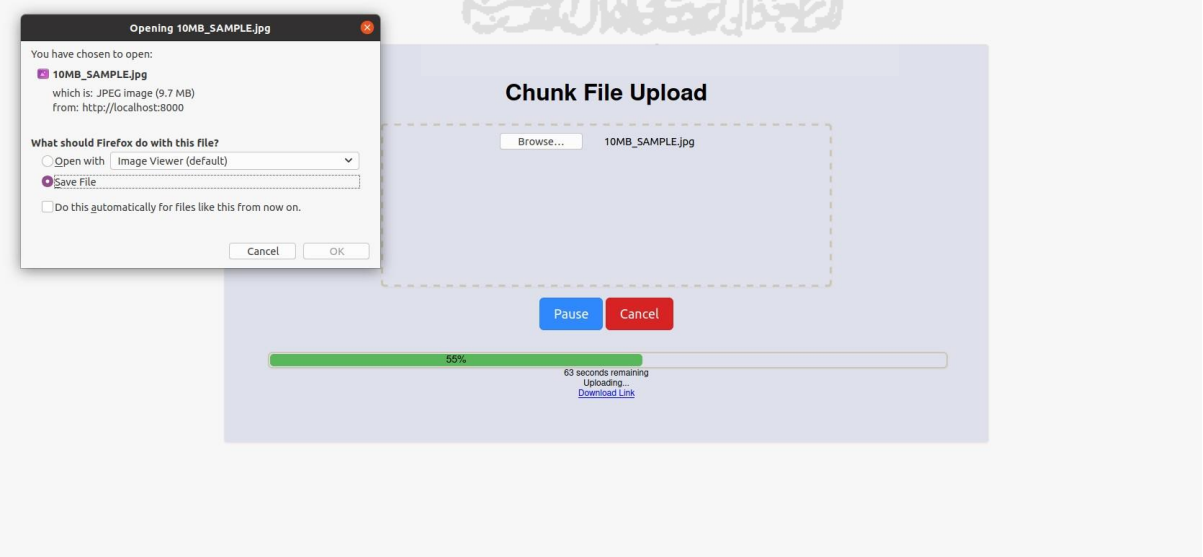
Gambar 4. 36 *Collection* "*fs.file*" pengujian ke-2.

Hasil yang didapatkan dari pengujian menggunakan file "10MB_SAMPLE", menunjukan bahwa setiap *chunked-file* telah disimpan dalam *database. Chunked-file* tersebut disimpan setelah *chunked-file* dari file "100MB_SAMPLE". Sama seperti pada pengujian dengan file "100MB_SAMPLE", *id-file* ini disimpan sebagai "*filename*" untuk menjadi pembeda antara *chunked-file* satu dengan *chunked-file* lainnya. Setelah sistem diuji menggunakan ukuran dan yang berbeda, dapat dilihat hasil dari kinerja sistem. Sistem dapat membuat setiap file menjadi file yang unik dan berbeda satu sama lain. Baik pada kelompok *chunked-file* yang sama, maupun antara *chunked-file* dalam satu kelompok. Hal ini membuat pengelolaan file dalam *database* menjadi lebih mudah. Untuk pengembangan pada tahap yang lebih besar, rancangan sistem ini dapat menjadi bahan pertimbangan. Karena dari segi penyimpanan dan pemrosesan file, sistem menunjukan kinerja yang sesuai dengan tujuan sistem dibuat. Setelah sistem diuji dalam hal pengiriman file hingga *database*, kemudian dilakukan pengujian untuk melihat ketersediaan file. Rangkaian pengujian selanjutnya dilakukan untuk menguji sistem dalam melayani proses unduh file. Karena sistem *repository* seperti ini difungsikan untuk melayani proses unggah dan unduh file. Pengujian ini juga menunjukan bahwa sistem dapat melayani proses pengunggahan tanpa meilhat format dari file yang akan diunggah. Dikarenakan chunking dilakukan dengan memotong isi dari setiap file sesuai dengan besaran dari file tersebut. Maka sistem tidak akan membaca fotmat maupun file signature dari file yang akan

diunggah. Sehingga file dengan segala format dapat dimasukan kedalam sistem untuk disimpan dalam database.

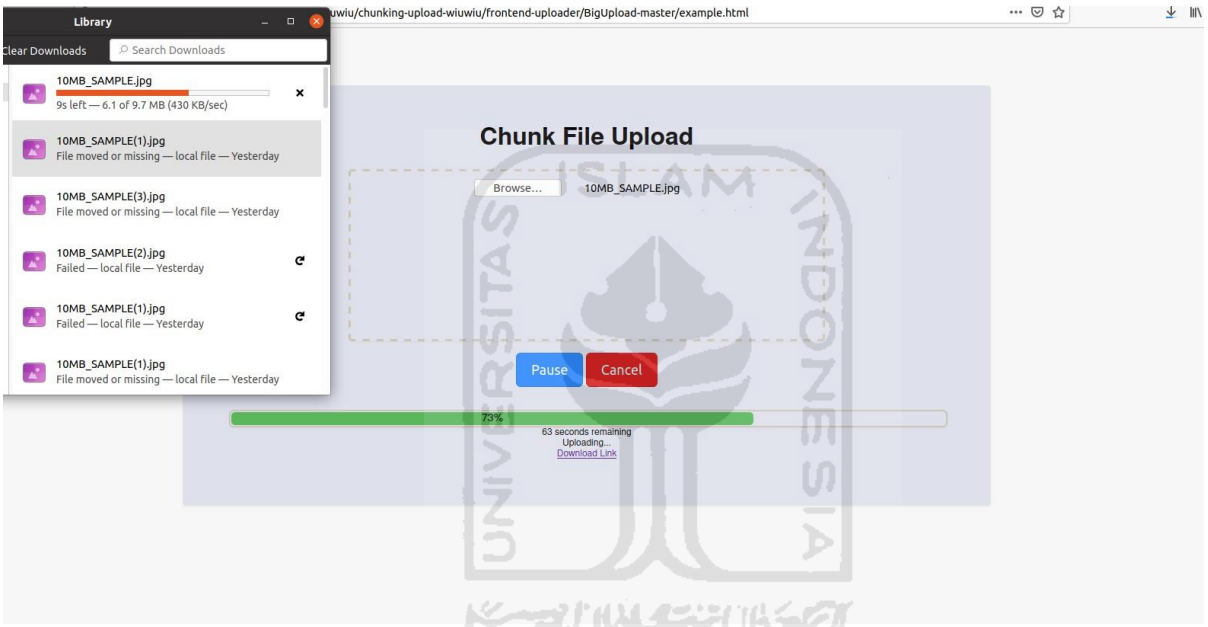
4.4.2 Ketersediaan File

Ketersediaan file merujuk pada aspek *high-availability* sebuah sistem, terutama pada bagian *database*. Dengan kata lain, menguji kinerja sistem dalam melayani *request* terhadap pengunduhan file. Dimana setiap komponen atau bagian dari sistem sangat berpengaruh pada kinerja sistem secara keseluruhan. Rangkaian pengujian ini dilakukan dengan menggunakan file "10MB_SAMPLE" dengan ekstensi *jpg*. Setelah *server-side* aktif dan terkoneksi dengan *database*, pengujian dilakukan dengan memulai pengunggahan file "10MB_SAMPLE". Ketika sistem sedang melakukan proses pengunggahan file, dilakukan *request* untuk melayani proses pengunduhan file. Sistem ini dirancang untuk dapat melakukan proses pengunggahan dan pengunduhan file dalam waktu yang bersamaan. Pertama sistem dibiarkan untuk melakukan proses pengunggahan hingga *progress-bar* menunjukan persentase tertentu. Kemudian secara acak dilakukan proses pengunduhan dengan menekan tombol "*Download Link*". Skenario pengujian ini dilakukan pada angka persentase yang acak atau tidak terukur. Setelah itu dilakukan pemantauan terhadap respon dari sistem dalam skenario ini. Gambar 4.34 adalah hasil yang ditunjukan sistem terhadap *request* pengunduhan file ketika proses pengunggahan berlangsung.



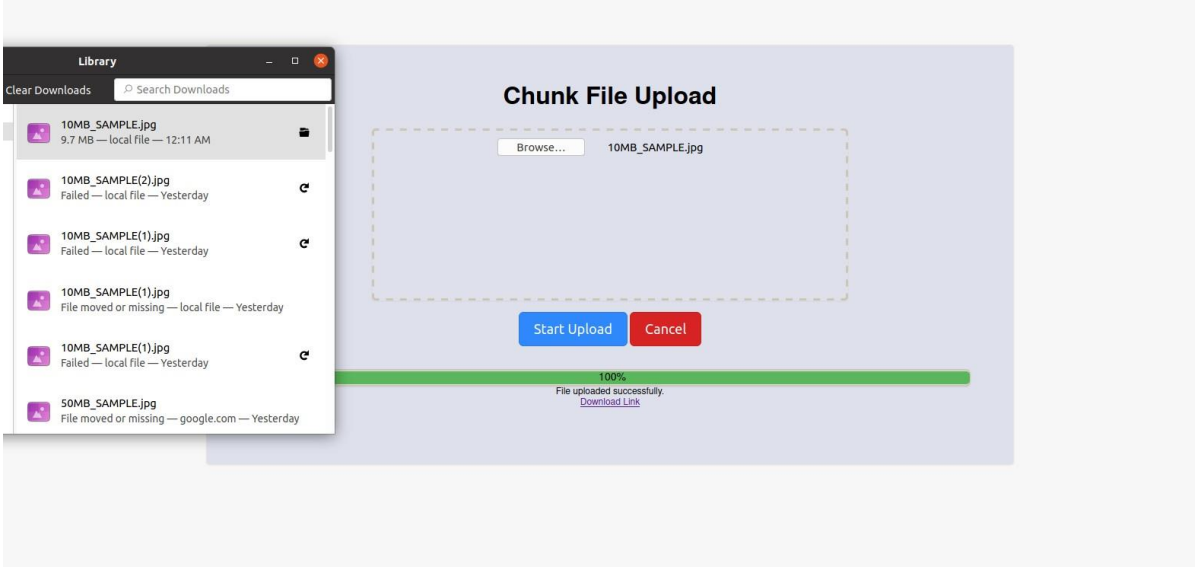
Gambar 4. 37 *Downloading file*.

Sistem melayani proses pengunduhan dari setiap *chunked-file* yang telah disimpan dalam *database*. Artinya sistem dapat melayani proses pengunduhan bersamaan dengan proses pengunggahan terhadap file yang sama. Apabila melihat gambar 4.37, sistem yang sedang melakukan proses pengunggahan pada angka 56% ini dapat melayani proses unduh. Dengan besaran dan nama file yang sama dengan file yang diunggah. Pengujian dilakukan dengan memulai proses pengunduhan file. Sistem diuji untuk dapat mengunduh file dengan lengkap tanpa ada kecacatan. Gambar 4.38 adalah proses unduh file dari *database* melalui aplikasi *web-browser*.



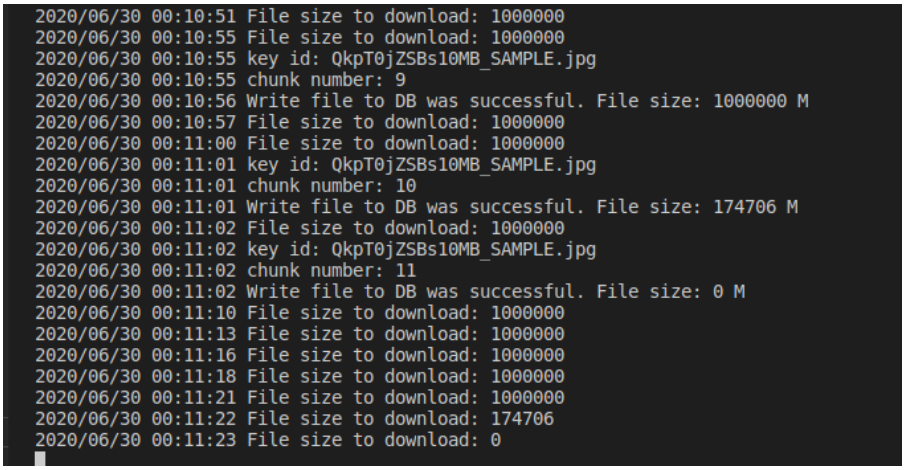
Gambar 4. 38 Proses pengunduhan file.

Proses unduh file terjadi bersamaan dengan proses pengunggahan yang hampir mencapai 100%. Namun skenario pengujian ini belum selesai, karena proses pengunduhan file bisa saja gagal walaupun proses pengunggahan file telah berhasil. Oleh karena itu, sistem dapat dikatakan berhasil apabila file yang diunduh telah benar-benar disimpan dalam direktori lokal. Gambar 4.39 adalah sistem ketika telah mencapai 100% untuk kedua proses yang berlangsung sebelumnya.



Gambar 4. 39 Pengunduhan file selesai

Proses pengunggahan file telah mencapai angka 100%, yang berarti keseluruhan file telah disimpan pada *database*. Untuk file "10MB_SAMPLE" yang sebelumnya telah dilakukan proses unduh, juga memperlihatkan hasil 100%. Artinya kedua proses tersebut telah mencapai 100% dalam waktu yang hampir bersamaan. Skenario pengujian ini membuktikan bahwa sistem dapat melayani 2 proses secara bersamaan. Walaupun secara teknis sistem melayani proses secara bergantian, namun dari perspektif pengguna sistem dapat berjalan dengan 2 proses. Skenario pengujian ini juga dapat menguji *server-side* sebagai bagian sistem yang menghubungkan *client-side* dan *database*. Gambar 4.40 adalah hasil yang ditunjukan ketika pengujian pada *server-side*.

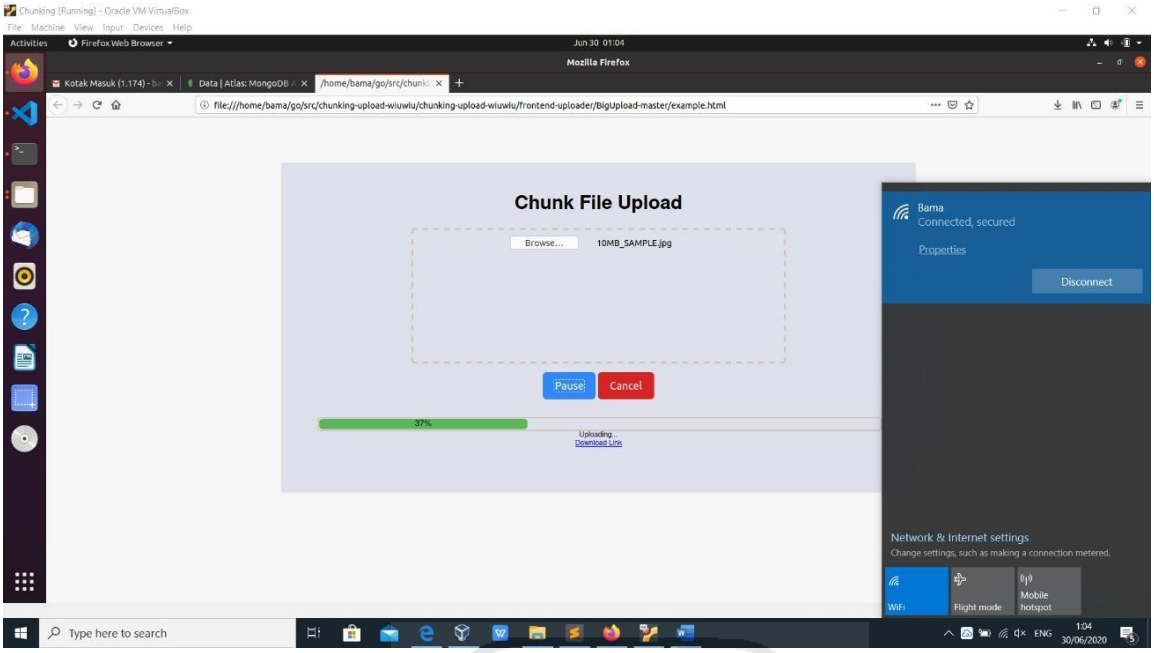


Gambar 4. 40 Proses pengunduhan pada *server-side*.

Gambar 4.40 memperlihatkan bahwa *server-side* sedang melakukan pengiriman file ke *database* namun di sela proses pengiriman, *server-side* juga melakukan pengunduhan file. *Server-side* melakukan proses pengiriman file secara berkala, yaitu dengan mengirimkan potongan demi potongan. Potongan file ini dikirim dengan ukuran 1 MB atau sama dengan ukuran *chuked-file* pada *database*. Dengan kata lain, *server-side* melakukan proses pengiriman file ke *client-side* sama dengan ketika melakukan proses pengiriman file ke *database*. Sehingga dapat dikatakan bahwa sistem memiliki tingkat ketersediaan file yang tinggi. Sistem dengan metode *chunking* ini juga dapat melayani proses unduh pada file yang semula telah dipotong-potong. Setelah skenario pengujian ini, dilakukan tahap pengujian tentang kehandalan sistem dalam melayani pengunggahan file dalam kondisi internet yang *unstable* dan *unreliable*.

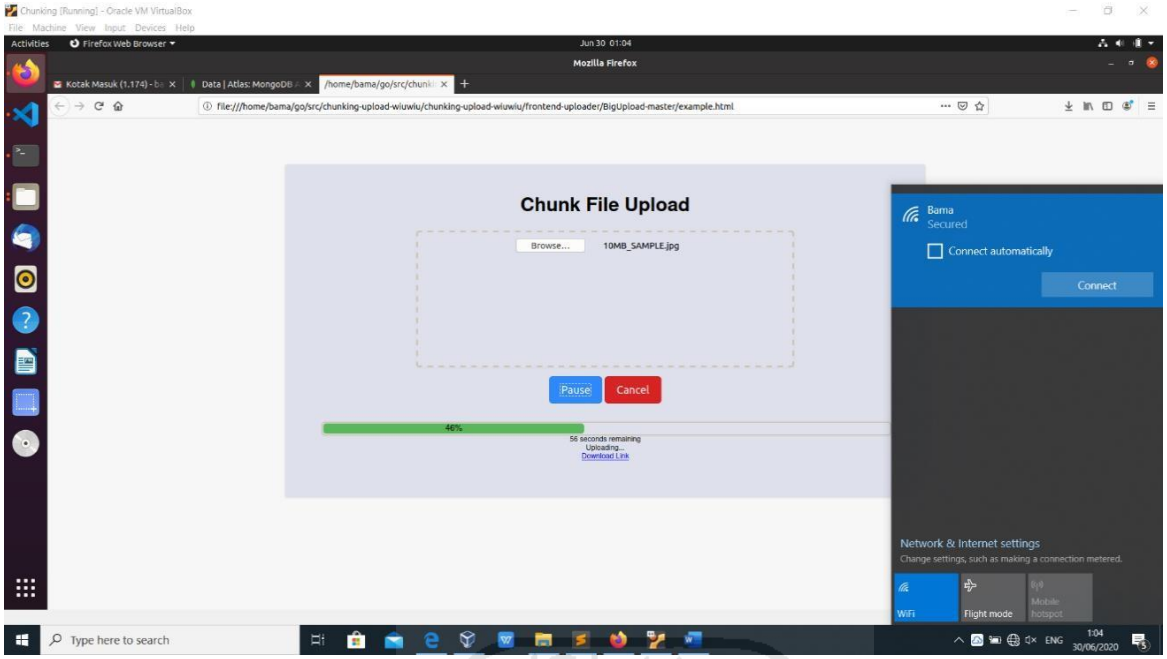
4.4.3 Kehandalan Sistem

Skenario pengujian ini dilakukan dengan memutuskan koneksi internet ketika proses pengunggahan file berlangsung. Karena lokasi *database* yang berada di North Viriginia, maka jika internet terputus pengiriman tidak dapat dilakukan. Skenario ini mewakili kebutuhan internet dalam proses pengunggahan file dari *client-side* menuju *database*. Pertama digunakan file "10MB_SAMPLE" dengan ekstensi jpg sebagai file yang akan dikirimkan. Ketika proses pengunggahan file "10MB_SAMPLE" ke *database* berlangsung, internet dimatikan. Proses terputusnya internet diwakili dengan memutuskan *wifi*, dimana dalam hal ini *wifi* digunakan peneliti sebagai konektivitas utama ke internet. Mula-mula sistem dibiarkan untuk melakukan proses pengunggahan file dan pemutusan koneksi internet dilakukan secara acak. Maksud dari secara acak adalah memutuskan internet pada persentase yang tidak ditentukan sebelumnya. Karena dalam kondisi internet yang tidak stabil, konektivitas internet dapat terputus kapanpun tanpa dapat diprediksi. Gambar 4.41 adalah hasil yang ditunjukan sistem dari Ketika skenario pengujian ini dilakukan.



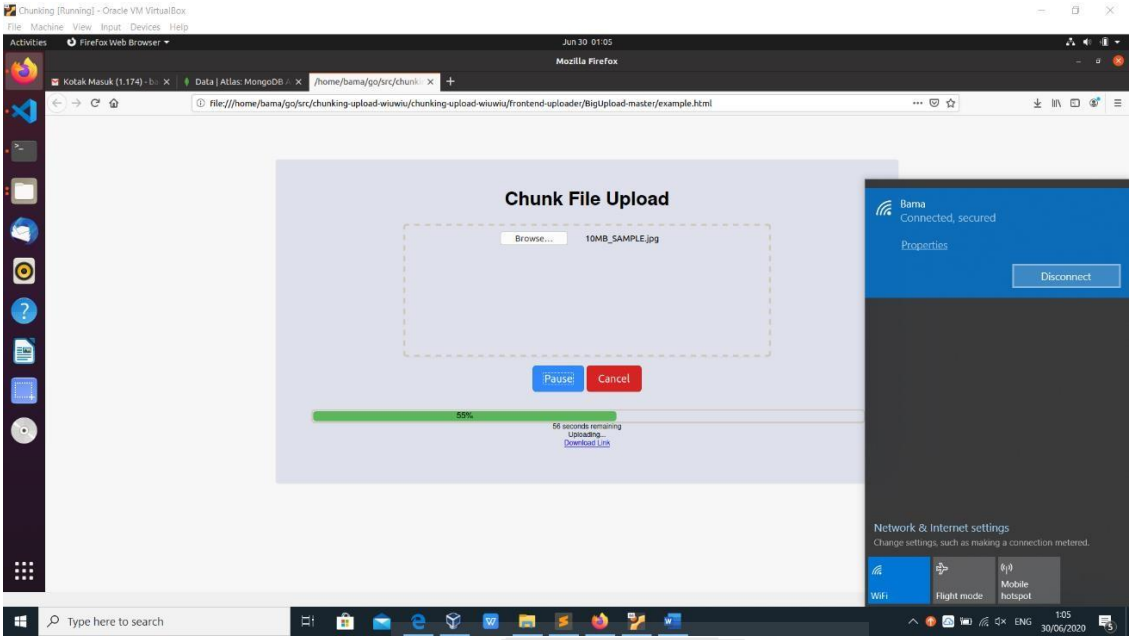
Gambar 4. 41 Proses pengujian kehandalan sistem.

Mula-mula sistem memperlihatkan persentase pengunggahan yang telah mencapai 37%. Kemudian pada persentase 44% koneksi internet diputus, sehingga sistem tidak dapat melanjutkan prosesnya hingga selesai. Sistem dirancang untuk dapat melanjutkan proses pengiriman ketika internet telah tersedia kembali. Dengan kata lain, sistem tidak mengulangi proses pengiriman dari persentase 0% seperti sistem konvensional lainnya. Rancangan ini digunakan untuk menciptakan sistem yang dapat digunakan dalam kondisi internet yang buruk. Untuk mewakili skenario tersebut, koneksi internet dimatikan agar sistem dapat menghentikan proses pengiriman file. Gambar 4.42 adalah skenario terputusnya koneksi internet.

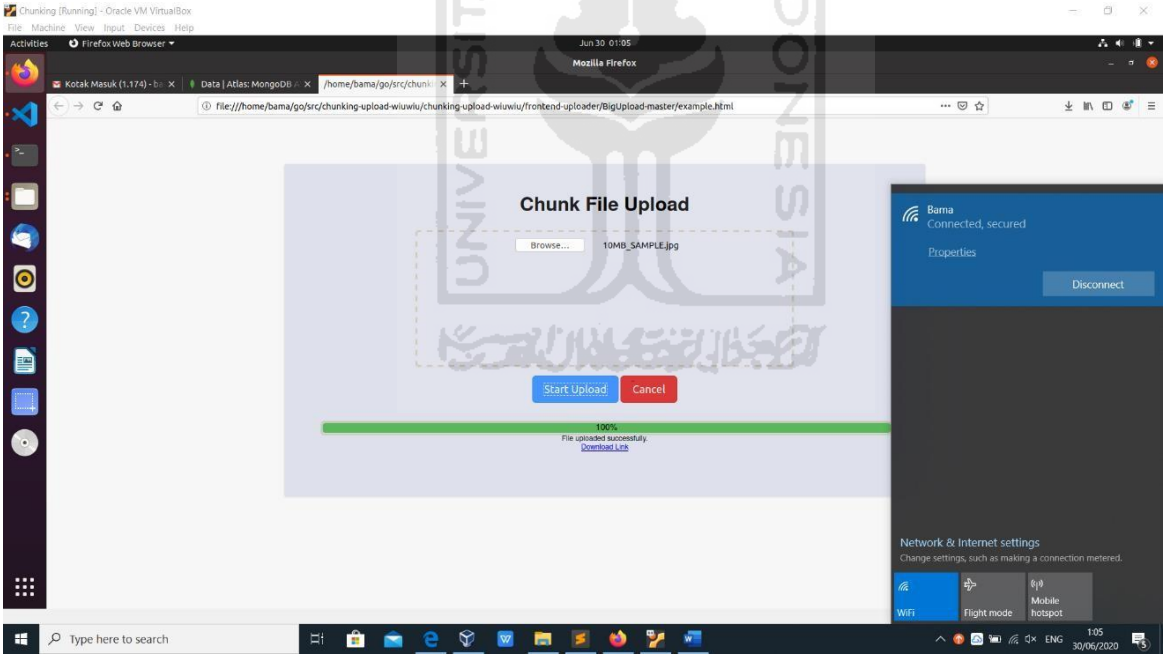


Gambar 4. 42 Proses pemutusan koneksi.

Setelah koneksi internet terputus, terlihat sistem menghentikan proses pengiriman filenya. Pada persentase 44%, sistem terhenti karena tidak dapat terhubung dengan *database* yang berlokasi di North Virginia. Skenario ini dilakukan untuk menguji kehandalan sistem dalam melayani proses pengiriman file. Setelah skenario internet terputus, dilakukan pengujian dengan menghubungkan kembali internet. Internet dihubungkan kembali dengan menyalakan *wifi*, dimana *wifi* adalah konektivitas utama dari pengujian ini. Gambar 4.43 merupakan skenario pengujian dengan menghubungkan kembali internet.



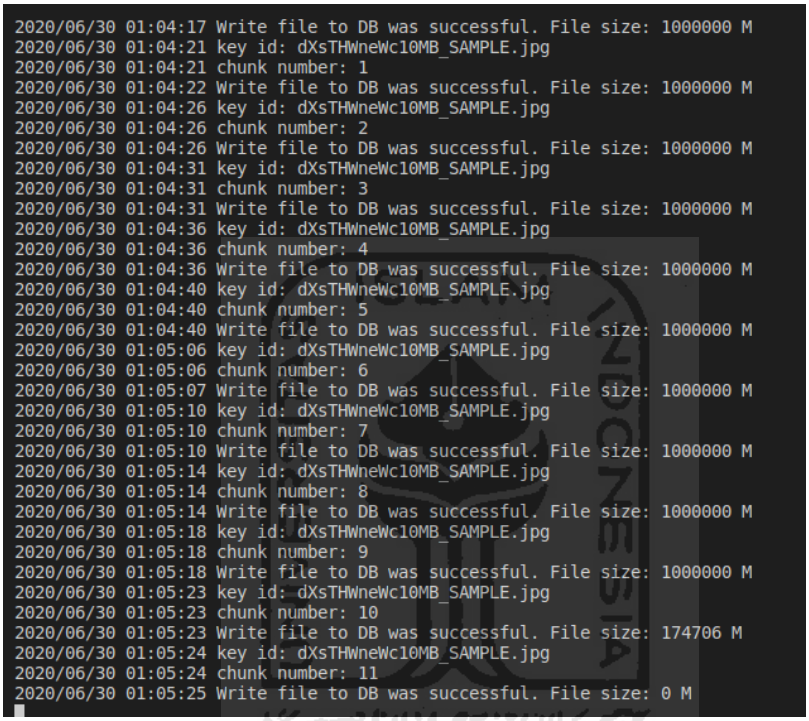
Gambar 4. 43 Proses pengunggahan dilanjutkan.



Gambar 4. 44 Pengunggahan selesai pengujian ke-4.

Setelah internet dihubungkan kembali, sistem menunjukan kembali progres pengiriman file dengan melanjutkan proses yang berjalan sebelumnya. Sistem dapat melanjutkan proses pengiriman hingga selesai, yang berarti sisa dari *chunked-file* yang belum dikirimkan telah disimpan dalam *database*. Kehandalan sistem ini diuji berdasarkan skenario yang mewakili internet terputus. Sehingga sistem yang dirancang haruslah dapat menyelesaikan proses

pengiriman file ketika internet terhubung kembali. *Server-side* sebagai penghubung antara *client-side* dan *database* memiliki peran penting dalam skenario pengujian ini. Karena *server-side* memiliki tugas untuk menyimpan file ke dalam *database*. Pada skenario pengujian ini *server-side* menjadi bagian yang dituntut untuk bekerja secara maksimal. Oleh karena itu dilakukan pengujian pada *server-side* dengan melihat proses yang terjadi didalamnya. Gambar 4.45 adalah proses yang terjadi selama skenario pengujian dilakukan.

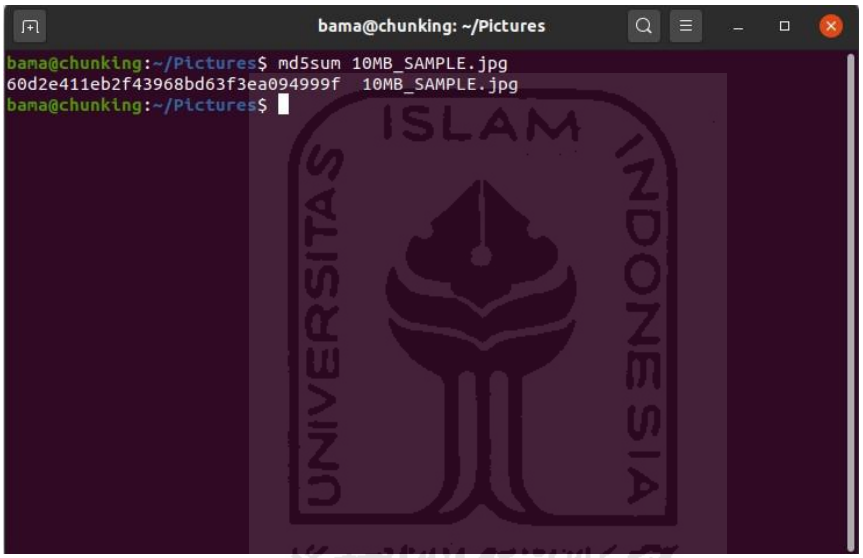


Gambar 4. 45 Proses pada *server-side* pengujian ke-4.

Server-side menunjukan proses pengiriman yang sama seperti pengiriman tanpa internet yang sempat terputus. Dengan mengirimkan *chunked-file* dari 1 hingga 11 ke *database* tanpa adanya kesalahan. Berdasarkan proses yang terjadi di *server-side* ketika skenario dijalankan, *server-side* telah menyimpan seluruh *chunked-file* ke *database*. Walaupun internet sempat terputus pada persentase 44%, *server-side* tetap mengirimkan 54% sisanya tanpa mengulangi dari 0%. Pengujian ini membuktikan bahwa sistem dapat digunakan dalam kondisi internet yang tidak memadai. Skenario pengujian ini juga membuktikan kehandalan sistem yang dirancang dalam menghadapi masalah internet tidak memadai.

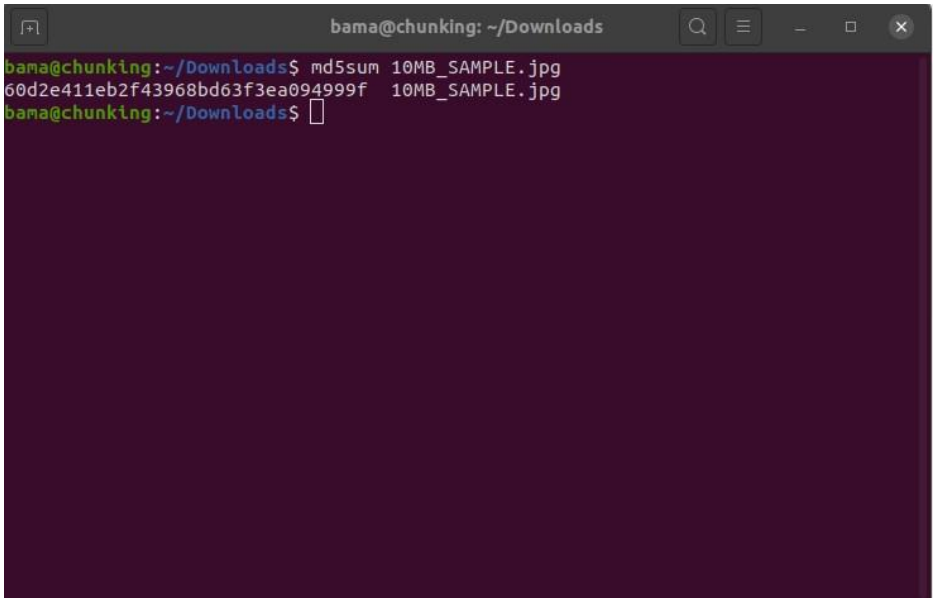
4.4.4 Pengecekan *hash* menggunakan MD5

Skenario pengujian ini dilakukan dengan mengamati nilai *hash* MD5 pada file yang diunggah dan file hasil pengunduhan. Proses pengujian dilakukan dengan memanfaatkan linux command yaitu “”md5sum”. Commad ini akan membuka *hash* md5 untuk setiap file dan menampilkannya pada command line. Pertama dilakukan pengecekan *hash* pada file sample “10MB_SAMPLE” yang digunakan dalam pengujian pengunggahan file. Command “md5sum” terhadap file tersebut akan menampilkan *hash* md5 yang menunjukan kode identitas dari file tersebut. Gambar 4.46 menunjukan *hash* pada file “10MB_SAMPLE” yang digunakan dalam pengunggahan file.



Gambar 4. 46 MD5 *hash* pada file yang diunggah.

Hash dengan MD5 menunjukan enkripsi dari kode untuk identitas file “10MB_SAMPLE”. Hasil dari pengujian pengecekan *hash* dengan MD5 pada file tersebut menujukan kode “60d2e11eb2f4396bd63fea094999f”. Ini membuktikan bahwa MD5 *hash* dari file yang diunggah sebelumnya memiliki kode identitas 60d2e11eb2f4396bd63fea094999f. Kemudian untuk menunjukan bahwa file hasil dari pengunduhan terhadap file yang sama adalah identik dengan file sebelumnya. Maka dilakukan pengecekan *hash* dengan MD5 pada file hasil pengunduhan. Gambar 4.47 memperlihatkan *hash* MD5 pada file hasil pengunduhan.



Gambar 4. 47 *Hash* MD5 file hasil pengunduhan.

File “10MB_SAMPLE” menunjukan *hash* MD5 yang serupa dengan file yang sebelumnya diunggah. Dengan kode enkripsi MD5 yaitu 60d2e11eb2f4396bd63fea094999f, yang berarti kedua file memiliki kode identitas yang identik dan serupa. Hal ini menunjukan bahwa file yang diunggah dan diunduh tidak mengalami kekurangan maupun penamahan untuk setiap nilai bit didalamnya. Dapat juga dikatakan bahwa file yang diunggah merupakan file yang diunduh tanpa adanya perbedaan. Perbedaan hanya dapat ditemukan dalam file signature, karena adanya perbedaan waktu dari penyimpanan kedua file. Pengujian ini membuktikan bahwa sistem pada penelitian ini dapat mengunggah dan mengunduh file tanpa adanya kerusakan dan perbedaan pada file tersebut. Sistem dengan metode *chunking* dapat diuji validitas terhadap setiap file yang disimpan didalam database file yang disimpan didalam *database*.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada penelitian ini telah disajikan sistem pengunggahan dan pengunduhan file menggunakan metode *chunking* dan memanfaatkan *unstructured database*. Sistem dirancang melalui proses yang telah sesuai dengan hasil dari tahapan analisis. Dari proses-proses yang telah dilakukan dalam penelitian, dapat diambil kesimpulan sebagai berikut:

a. Sistem dapat melakukan pengunggahan file menggunakan metode *chunking*.
b. *Client-side* dapat memuat antarmuka yang merepresentasikan proses yang sedang terjadi pada sistem.
c. Sistem dapat melakukan proses pengunduhan file sesuai dengan file yang semula (sebelum melalui proses *chunking*).
d. Sistem dengan metode *chunking* dapat digunakan dalam kondisi internet yang *unstable* dan *unreliable*.
e. Ketika internet terhubung kembali setelah sempat terputus, sistem dapat melanjutkan proses pengunggahan file dimulai dari *chunked-file* terakhir yang terhenti.
f. Sistem dapat menyimpan *chunked-file* di *database*.
g. Sistem dapat memberikan *filename* yang unik untuk setiap *chunked-file* yang disimpan di *database*.
h. *Unstructured database* dapat digunakan untuk menyimpan file hasil proses *chunking*.
i. *Unstructured database* dapat melakukan replikasi file dari *server*.
j. MongoDB Atlas sebagai *unstructured database* dapat terkoneksi dengan *server* yang menggunakan bahasa pemrograman Golang.

5.2 Saran

Pada bagian akhir penelitian ini, peneliti mengajukan saran untuk pengembangan pada penelitian yang memiliki lingkup lebih luas. Masih banyaknya hal-hal yang perlu dikembangkan agar penelitian ini dapat berkembang lebih sempurna pada penelitian selanjutnya. Saran yang diajukan adalah sebagai berikut:

a. Mengimplementasikan metode *chunking* pada sistem dengan skala yang lebih besar. Sebagai contoh, sistem perkuliahan yang menyediakan penyimpanan berkas untuk mahasiswa.

b. Mencari metode pengunggahan file yang lain, kemudian dibandingkan dengan metode *chunking*.
c. Menambah sistem keamanan untuk setiap *chunked-file*.
d. Menggunakan *database* dengan *storage* yang lebih besar.
e. Menggunakan *device*/komputer dengan spesifikasi yang lebih mumpuni agar mencapai kelancaran dalam penelitian.
f. Memperbaiki tampilan dari sistem agar lebih menarik dan menyenangkan untuk digunakan.

DAFTAR PUSTAKA

Asosiasi Penyelenggara Jasa Internet Indonesia. (2018). *Penetrasi & Profil Perilaku Pengguna Internet Indonesia.* Indonesia: APJII.

Berggren, E. A. (2017). A Comparison Between MongoDB and MySQ LDocument Store Considering Performance. 1-10.

Chieh Ming Wu, Y. F. (2015). Comparisons Between MongoDB and MS-SQL Databases on the TWC Website. *American Journal of Software Engineering and Applications* , 35-41.

Dildar Husain, M. O. (2019). Load Status Evaluation For Load Balancing In Distributed Database Servers. *3C Tecnología. Glosas de innovación aplicadas a la pyme*, 427-440.

Gibson, K. R. (2015). TABLEFS: Embedding a NoSQL Database Inside the Local File System . 1-6.

Ilker Nadi Bozkurt, A. A. (2017). Why Is the Internet so Slow?! *Springer International Publishing AG* , 174-180.

International Telecommunication Union. (2019, December 20). *Statistics*. Diambil kembali dari www.itu.int: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

Mohamed-Amine Baazizi, G. G. (2019). Schemas And Types For JSON Data. *open proceedings*, 438-441.

Mostafa R. Kaseba, M. H. (2019). An improved technique for increasing availability in BigData replication. *Future Generation Computer Systems*, 493-503.

Peter Kunszt, E. L. (2005). File-based Replica Management. *Future Generation Computer System*, 1-9.

Ryan N.S. Widodo, H. L. (2017). A new content-defined chunking algorithm for data deduplication in cloud storage. *Future Generation Computer System*, 145-153.

Srikanth Sundaresan, W. d. (2013). Broadband Internet Performance: A View From the Gateway . *HAL*, 1-6.

Thanh Trung Nguyen, T. K. (2015). BFC: High-Performance Distributed Big-File Cloud Storage Based On Key-Value Store. *VNG Research*, 1-7.

Youjip Won, K. L. (2014). MUCH: Multithreaded Content-Based File Chunking . 1-6.

LAMPIRAN