STUDI KELAYAKAN RINCI PEMBANGUNAN SISTEM *WEB SERVICES* SEBAGAI CARA KOMUNIKASI DENGAN TEMPAT PEMBAYARAN DALAM PENCATATAN PEMBAYARAN PAJAK BUMI DAN BANGUNAN PERDESAAN DAN PERKOTAAN DI KABUPATEN BREBES.

29 Agustus 2016

Priyanto Tamami, S.Kom.

1 RUANG LINGKUP PEKERJAAN

Ruang lingkup dari pekerjaan membangun sistem *web services* sebagai bentuk atau cara komunikasi dengan tempat pembayaran dalam pencatatan pembayaran PBB-P2 di Kabupaten Brebes sebetulnya sudah cukup jelas, yaitu membangun sebuah layanan (atau dalam bahasa Teknologi Informasi adalah *server*) yang menyediakan skema *web services* namun bentuk yang lebih jelas dari penggunaan *web services* disini akan berbentuk arsitektur *Rest* dan hasil yang dikeluarkan oleh sistem aplikasi ini akan berformat JavaScript Object Notation (JSON).

Server nantinya hanya akan memberikan 3 (tiga) layanan standar yaitu : *inquiry* untuk pengambilan informasi mengenai Objek Pajak maupun Subjek Pajak, pencatatan pembayaran, dan pembatalan pencatatan pembayaran.

Maksud dari ketiga layanan tersebut adalah sebagai berikut :

1. *Inquiry*

   Layanan ini digunakan apabila Bank sebagai tempat pembayaran ingin mengambil informasi mengenai objek pajak dan wajib pajak, untuk memastikan bahwa memang ada data dengan nomor objek tersebut, dapat pula

digunakan sebagai bahan untuk verifikasi data, yaitu untuk memastikan apakah dengan nomor objek pajak tersebut, nama wajib pajak dan besarnya pajak bumi dan bangunan yang terhutang sesuai dengan yang tertera di dalam Surat Pemberitahuan Objek Pajak atau pernyataan dari Wajib Pajak / Kuasanya dalam membayar.

2. Pencatatan Pembayaran

   Layanan ini digunakan oleh Bank sebagai tempat pembayaran untuk melakukan pencatatan pembayaran pada basis data yang berada di DPPK Kabupaten Brebes. Layanan ini akan otomatis melakukan pencatatan pembayaran pada basis data SISMIOP dengan tentu saja memberikan respon berupa informasi nomor objek mana yang dicatatkan pembayarannya, atas nama siapa pajak bumi dan bangunan terbayarkan, dan berapa jumlah tagihan yang terbayar.

3. Pembatalan Pembayaran

   Layanan ini digunakan oleh Bank sebagai tempat pembayaran untuk melakukan pembatalan pembayaran apabila proses pencatatan pembayaran yang diminta oleh Bank sebagai tempat pembayaran telah mencapai batas waktu *timeout* yang telah ditetapkan oleh Bank tetapi layanan *web services* yang berada di DPPK belum memberikan respon. Atau ada kondisi lain yang menyebabkan pencatatan pembayaran harus dibatalkan.

Hal-hal yang mendukung studi kelayakan dibangunnya sistem *web services* untuk pencatatan pembayaran PBB-P2 adalah dari kondisi-kondisi berikut :

1. Wawancara

   Hal ini dilakukan guna memperkuat kebutuhan akan adanya layanan pencatatan pembayaran yang akurat dan dengan data terkini (atau dengan istilah

lain adalah *realtime*). Ini dibutuhkan terutama untuk pengambilan keputusan atau kebijakan bila misalkan suatu saat dilakukan kegiatan intensifikasi ke sebuah Desa / Kecamatan, akan terlihat efektif bila dalam masa beberapa waktu setelah kegiatan intensifikasi, seharusnya ada peningkatan realisasi pembayaran untuk Desa / Kecamatan tersebut.

Dari hasil wawancara maka dapat disimpulkan bahwa beberapa keinginan yang diharapkan dari sistem pencatatan *realtime* ini (atau dikenal dengan sistem *host-to-host*), adalah sebagai berikut :

(a) Tersedianya sumber data yang akurat dan konsisten untuk menunjukkan piutang dan realisasi.

(b) Dapat tersedianya realisasi pembayaran waktu nyata atau *realtime* guna pengambilan keputusan / kebijakan.

(c) Pembayaran yang tercatat langsung secara waktu nyata atau *realtime* dapat digunakan sebagai informasi pertanggungjawaban langsung kepada wajib pajak atau kuasanya dalam melakukan kontrol terhadap pembayaran-pembayaran yang dilakukan melalui petugas pemungut.

2. Pengumpulan Bukti Pendukung

Data pendukung lain untuk memperkuat asumsi-asumsi atau pendapat dari hasil wawancara adalah dengan pengumpulan bukti-bukti yang dapat disimpulkan sebagai berikut :

(a) Bahwa data antara SISMIOP (basis data yang berada di DPPK), dengan Cash Management System (CMS) yaitu basis data yang berada di Bank sebagai tempat pembayaran, ada selisih baik kondisi pada jumlah objek piutang dan besarnya nilai piutang, serta jumlah objek pembayaran dan besarnya nilai pembayaran yang tercatat.

(b) Bahwa masih adanya penyalahgunaan kewenangan petugas pemungut dalam melakukan pemungutan PBB-P2 dan penyetoran hasil pemungutannya ke Bank sebagai tempat pembayaran.

(c) Bahwa masih adanya selisih waktu pencatatan pembayaran antara basis data yang berada di DPPK dan Bank sebagai tempat pembayaran, yang ini tidak dapat diubah karena menjadi keterbatasan sistem bahwa data-data nomor objek pajak yang telah dibayarkan baru dapat diambil dan dipindahkan ke sistem basis data SISMIOP di DPPK setelah melalui waktu *cut-off* yang ditetapkan Bank sebagai tempat pembayaran.

Dari data-data tersebut, maka sudah seharusnya bentuk sistem *host-to-host* dibangun, karena bukan hanya akan mendapatkan nilai realisasi pembayaran dalam waktu nyata (*realtime*) melainkan konsistensi data yang terjaga dalam satu basis data. Sebagai nilai lebih adalah dapat digunakan sebagai sebuah alat atau media pertanggungjawaban petugas pemungut PBB-P2 dalam melakukan penyetoran hasil pemungutannya ke Bank sebagai tempat pembayaran.

2 SARANA DAN PRASARANA YANG MELIPUTI PERANGKAT KERAS DAN PERANGKAT LUNAK YANG DIPERLUKAN

Sebagai sebuah sistem yang utuh, maka diperlukan subsistem pembentuknya agar keseluruhan sistem dapat bekerja dengan baik. Tentu karena yang dibangun adalah sebuah sistem layanan (*service*) yang menggunakan teknologi informasi, maka tidak akan lepas dari kebutuhan perangkat lunak dan perangkat keras yang

mendukungnya. Kebutuhan akan perangkat lunak dan perangkat keras tersebut disebutkan sebagai berikut :

1. **Perangkat Lunak**

   Perangkat lunak sebagai pembentuk sistem layanan *web services* ini berjalan adalah sebagai berikut :

   (a) Basis Data

   Sebagaimana sebuah sistem yang melakukan pengolahan data, maka diperlukan tempat untuk menyimpan data-data tersebut yang biasa dikenal dengan basis data (*database*). Karena sistem layanan *web services* ini akan mencatatkan langsung permintaan sebuah transaksi pembayaran, maka basis data yang digunakan oleh sistem layanan *web services* ini sama dengan basis data yang digunakan dalam pengelolaan objek dan subjek pajak PBB-P2, yaitu SISMIOP, sistem basis data ini menggunakan sistem basis data Oracle 11g yang tentu saja berada di DPPK Kabupaten Brebes. Hanya saja, dalam pengoperasiannya, perlu adanya perubahan struktur basis data di SISMIOP atau lebih tepatnya penambahan tabel dan prosedur dalam basis data SISMIOP agar layanan *web services* dapat mencatatkan transaksi pembayaran lebih cepat dan historis dari kegiatan layanan *web services* ini tercatat.

   (b) *Text Editor*

   Tentunya untuk membangun sebuah sistem aplikasi yang terdiri dari susunan kode-kode program, dibutuhkan *text editor* atau *Integrated Development Environment* (*IDE*), untuk penggunaan *text editor* sendiri banyak yang menawarkan secara gratis, mulai dari vi, vim, emacs, notepad, atau perangkat lunak lain yang bertugas sebagai *text editor* pasti dapat digunakan sebagai salah satu alat untuk membangun sistem

aplikasi ini. Untuk IDE adalah sebuah perangkat lunak dengan dukungan fasilitas yang lebih lengkap dibandingkan sebuah *text editor* biasa, hal umum yang biasa ditawarkan oleh sebuah IDE adalah *highligth* dari kode-kode yang ditulis sehingga memudahkan dalam pembacaan kode program, bahkan ada yang menyertakan fasilitas *code snippets* sehingga seorang pemrogram tidak perlu terlalu sering untuk melihat dokumentasi sebuah modul karena secara otomatis, IDE akan memberikan informasinya seketika saat pemrogram kebingungan tentang apa nama fungsi atau prosedur yang akan digunakan.

IDE ini pun banyak yang menyediakan secara gratis dan selalu diperbaharui secara berkala. Salah satunya adalah Eclipse.

(c) Pustaka (*Library*)

Pustaka atau *library* adalah modul modul pendukung terbangunnya sebuah sistem aplikasi layanan *web services* ini. Pustaka atau *library* yang dibutuhkan sistem aplikasi ini cukup menggunakan yang gratis, namun tentu saja, handal dalam melaksanakan tugasnya.

Sebagai inti yang membangun sistem aplikasi ini, nantinya akan menggunakan *framework* Spring, Spring ini merupakan *framework* yang digunakan untuk membangun sebuah aplikasi *enterprise*, termasuk *framework* yang ringan untuk mendukung secara penuh dalam pengembangan aplikasi *enterprise*.

Spring dapat digunakan untuk melakukan pengaturan deklarasi manajemen transaksi, *remote access* dengan menggunakan RMI, atau layanan web lainnya. Spring juga memungkinkan untuk menggunakan hanya modul-modul tertentu sehingga tidak perlu menggunakan semua modul spring dalam aplikasi apabila tidak diperlukan.

Adapun fitur yang ditawarkan oleh *framework* spring adalah sebagai

berikut :

- *Transaction Management*, dimana Spring *framework* menyediakan sebuah lapisan / *layer* abstrak yang umum untuk manajemen transaksi, sehingga memudahkan *developer* dalam melakukan manajemen transaksi.
- JDBC *Exception Handling* : yaitu lapisan / *layer* abstrak JDBC yang menawarkan *exception* yang bersifat hierarki sehingga memudahkan penanganan kesalahan.
- Spring menawarkan layanan integrasi yang terbaik dengan Hibernate, JDO, dan iBatis.
- MVC *Framework* : dimana Spring hadir dengan *framework* aplikasi web MVC yang dibangun di atas inti Spring. Spring merupakan *framework* yang sangat fleksibel dalam pengaturan strategi *interface*, dan mengakomodasi beberapa teknologi *view* seperti JSP, Velocity, Tiles, iText, dan POI.

Arsitektur dari *framework* Spring ini terbagi menjadi beberapa bagian, yaitu sebagai berikut :

1) Spring Core Container

   Spring *Core Container* terdiri dari modul-modul *spring-core*, *spring-beans*, *spring-context*, *spring-context-support*, dan *spring-expression*. Modul *spring-core* dan *spring-beans* adalah dasar dari *framework* Spring ini, termasuk di dalamnya adalah fitur IoC dan *Dependency Injection*. *BeanFactory* adalah implementasi paling mutakhir dari pola bawaanya. Kondisi ini menghapuskan kebutuhan pembuatan kerangka program dan menjadikan pemisahan antara ketergantungan konfigurasi dan spesifikasi dari logika program sesungguhnya.

Modul *spring-context* digunakan untuk mengakses objek dengan cara *framework* yang mirip dengan cara JNDI *registry*. Sedangkan modul *spring-context-support* memberikan dukungan integrasi pada pustaka / *library* pihak ke-3 untuk berjalan pada aplikasi Spring untuk *caching* (seperti menggunakan EhCache, Guava, JCache), *mailing* (seperti JavaMail), *scheduling* (seperti CommonJ dan Quartz), dan *template engines* (seperti FreeMarker, JasperReports, Velocity). Sedangkan modul *spring-expressiong* memberikan kemampuan handal dalam bahasa Expression untuk melakukan *query* dan manipulasi objek grafik saat *runtime*.

2) Spring AOP dan Instrumentasi

   Modul *spring-aop* memberikan kemampuan untuk melakukan AOP (Aspect Oriented Programming) yang implementasinya memungkinkan untuk mendefinisikan *method interceptor* dan *pointcut* untuk memperjelas pemisahan kode yang mengimplementasikan fungsionalitas yang seharusnya dipisahkan.

   Secara terpisah modul *spring-aspects* memberikan integrasi terhadap AspectJ.

   Sedangkan modul *spring-instrument* memberikan dukungan instrumen yang dapat digunakan dibeberapa aplikasi *server*.

3) Spring Messaging

   Sejak *Framework* Spring 4, terdapat modul *spring-messaging* dengan abstraksi kunci dari *Spring Integration* seperti *Message*, *MessageChannel*, *MessageHandler*, dan lainnya untuk memberikan layanan bagi aplikasi yang berbasis pesan (*messaging*). Modul ini juga menyertakan beberapa anotasi untuk memetakan pesan ke *method*, mirip seperti model pemrograman anotasi di Spring MVC.

4) Spring Data Access / Integration

Lapisan *Data access / Integration* terdiri dari modul modul JDBC, ORM, OXM, JMLS, dan *Transaction*.

Modul *spring-jdbc* menyediakan lapisan abstraksi JDBC yang menghilangkan *coding* membosankan dari JDBC yang menguraikan kode error dari tiap sistem basis data yang berbeda.

Modul *spring-tx* mendukung manajemen transaksi terprogram dan deklaratif untuk kelas-kelas yang mengimplementasikan *interface* khusus, dan semua POJO (*Plain Old Java Objects*.

Modul *spring-orm* memberikan dukungan lapisan integrasi dari API pemetaan objek relasional, termasuk diantaranya JPA, JDO, dan Hibernate. Modul *spring-orm* dapat digunakan untuk seluruh *framework* pemetaan objek relasional yang dikombinasikan dengan seluruh fitur yang Spring sediakan seperti salah satu contohnya adalah fitur manajemen transaksi deklaratif sederhana yang disebutkan diatas.

Modul *spring-oxm* memberikan lapisan abstraksi yang mendukung implementasi pemetaan Objek / XML seperti JAXB, Castor, XMLBeans, JiBX, dan XStream.

Dan modul *spring-jms* (*Java Messaging Service*) terdiri dari fitur untuk membuat dan mengolah pesan. Sejak Spring Framework 4.1, modul ini tergabung dalam *spring-messaging*.

5) Spring Web

Lapisan *Web* terdiri dari modul *spring-web*, *spring-webmvc*, *spring-websocket*, dan *spring-webmvc-portlet*.

Modul *spring-web* menyediakan basis integrasi *web-oriented* seperti fungsionalitas pengunggahan *multipart* dan inisialisasi dari IoC *con-*

tainer menggunakan Servlet *listener* dan *web-oriented application context*. Modul ini juga memuat klien HTTP dan bagian-bagian web dengan dukungan Spring *remote*.

Modul *spring-webmvc* (atau biasa dikenal dengan modul *Web-Servlet*) terdiri dari Spring *model-view-controller* (MVC) dan implementasi REST Web Services untuk aplikasi web. *Framework* Spring MVC memberikan pemisahan yang jelas antara domain model dan tampilan, juga mengintegrasikan dua hal tersebut dengan fitur *framework* Spring yang lain.

Modul *spring-webmvc-portlet* (atau biasa dikenal dengan modul *Web-Portlet*) menyediakan implementasi MVC yang digunakan pada lingkungan Portlet dan mewarisi fungsionalitas dari modul *spring-webmvc*.

6) Spring Test

Modul *spring-test* mendukung test unit dan test integrasi dari komponen-komponen Spring dengan JUnit atau TestNG. Modul ini memberikan pemuatan konsisten dari Spring *ApplicationContext*, dan melakukan *cache* dari konteks itu. Modul ini pun menyediakan objek tiruan yang dapat digunakan untuk melakukan tes dalam logika terisolasi.

(d) *Servlet Container*

Sebelum membahas apa itu *Servlet Container*, perlu diketahui terlebih dahulu apa itu *Web Server*. Dalam beberapa pengertian, *web server* menggunakan protokol HTTP untuk bertukar data. Dengan pengertian yang lebih sederhana, pengguna cukup mengetikan sebuah alamat *web* di *browser* (di sisi *client*), dan mengambil halaman *web* yang akan diakses atau dibaca. Jadi yang dilakukan *server* hanyalah mengirimkan

sebuah halaman *web* ke *client*. Perubahan hanya pada protokol HTTP yang menggunakan format spesifik dalam mengirim dan menerima pesan.

Lalu apa itu *servlet container*? Sebagaimana dijelaskan diatas bahwa pengguna / *client* hanya dapat meminta halaman *web* yang statis dari *server*. Hal ini kurang begitu bagus jika pengguna ingin membaca halaman *web* hanya berdasarkan input yang demikian. Sehingga ide dasarnya dari *servlet container* adalah menggunakan Java untuk membentuk halaman *web* di sisi *server* secara dinamis. Jadi sebetulnya, *servlet container* adalah bagian penting dari sebuah *web server* yang berinteraksi dengan *servlet*.

Pertanyaan berikutnya tentu saja, apa itu *servlet*? *Servlet* sendiri sebetulnya hanyalah sebuah *interface* dengan 3 (tiga) *method* yang menggambarkan siklus hidup sebuah *servlet*, yaitu *init()*, *service()*, dan *destroy()*. Tiap *method* dari *Interface* ini diimplementasikan pada kelas turunannya dan dipanggil oleh *server* pada waktu-waktu tertentu seperti penjelasan berikut :

i. *Method init()* dipanggil saat tahap inisialisasi dari siklus hidup *servlet*. *Method* ini mengirimkan objek yang mengimplementasikan *interface* javax.servlet.ServletConfig, yang memungkinkan *servlet* mengakses parameter inisialisasi dari aplikasi web.

ii. *Method service()* dipanggil setiap ada *request* setelah tahap inisialisasi. Setiap *request* dilayani dalam tiap *thread* masing-masing. *Container web* memanggil *method service()* dari *servlet* setiap ada *request*. *Method service()* menentukan jenis *request* yang ada dan mengirimkannya ke *method* yang akan menangani *request* tersebut.

iii. *Method desroy()* dipanggil ketika objek *servlet* harus dimusnakan.

Method ini akan melepaskan semua *resource* yang sedang digunakan.

Beberapa *servlet container* yang dikenal sebagai proyek *open source* adalah sebagai berikut :

- Apacha Tomcat (dahulu namanya Jakarta Tomcat).
- Apache Geronimo
- GlassFish
- JBoss
- Jetty

Sedangkan beberapa proyek komersial dari *servlet container* ini adalah sebagai berikut :

- JBoss Enterprise Application Platform
- JRun
- WebLogic Application Server
- Resin Pro
- IBM WebSphere Application Server.

Untuk kebutuhan, menggunakan Apache Tomcat yang bersifat gratis dan *open source* dirasa cukup untuk menangani aplikasi *web services* yang akan dibangun.

2. **Perangkat Keras**

   Perangkat keras yang membentuk sistem layanan *web services* ini berjalan adalah sebagai berikut :

   (a) Server Basis Data

Sebuah sistem basis data atau biasa dikenal perangkat lunak basis data, tentu saja harus berjalan diatas sistem operasi dimana sistem operasi pun memerlukan sebuah perangkat keras berupa komputer untuk dapat beroperasi. Karena kebutuhan dari sistem basis data ini untuk menyimpan dan mencatat data-data mengenai objek dan subjek pajak PBB-P2 dan termasuk melayani pencatatan pembayaran yang dilakukan dalam 24 jam sehari, maka diperlukan perangkat keras (*hardware*) berupa *server*. Server ini pun telah dimiliki, karena kegiatan operasi manajemen objek pajak yang dilakukan melalui perangkat lunak SISMIOP telah terhubung ke *server* tempat basis data ini, sehingga tidak perlu biaya tambahan lagi untuk perangkat keras ini.

(b) Server Aplikasi

Kondisi aplikasi dalam hal ini *servlet container* atau yang biasa dikenal *web server*, perlu ditempatkan dalam *server* terpisah, melihat kondisi dari *server* basis data yang telah melakukan tugasnya untuk menangani urusan data selama 24 sehari. Tujuannya agar kedua sistem aplikasi ini, yaitu sistem basis data dan sistem *web server* tidak saling memperebutkan *resouce* dari sebuah *server* yang pada akhirnya layanan akan terkendala lambatnya proses respon dari *server*.

Server ini pun sudah tersedia, sehingga tidak perlu biaya tambahan untuk pengadaan perangkat keras ini. *Server* sebelumnya ditujukan sebagai *server* aplikasi tempat aplikasi SISMIOP disimpan, sehingga komputer *client* perlu melakukan *remote* terlebih dahulu untuk menggunakan aplikasi SISMIOP ini. Namun karena terbentur masalah lisensi *remote desktop* sehingga *server* ini tidak digunakan sebagai *server* aplikasi, melainkan sebagai *server* uji coba dalam pembuatan sistem aplikasi, atau sebagai tempat untuk melakukan uji coba melakukan modi-

fikasi pada aplikasi SISMIOP, termasuk melakukan cetak massal melalui teknologi virtualisasi. Sementara aplikasi SISMIOP dipasangkan pada tiap komputer personal untuk melakukan manipulasi atau perubahan-perubahan data dalam konteks pelayanan PBB-P2.

(c) Router

Router digunakan untuk menghubungkan antara jaringan internet dengan jaringan lokal di DPPK, agar *server* aplikasi dapat dihubungi atau diakses dari internet, maka dibutuhkan *router* untuk menerjemahkan, alamat IP mana di jaringan lokal yang akan dihubungi apabila ada *request* dari jaringan internet. Ini adalah salah satu fungsi dari *router*, masih banyak fungsi lainnya seperti menyeleksi atau memfilter *port* mana saja yang dapat diakses dari internet, dan berapa lebar pita (*bandwidth*) yang akan dibatasi.

(d) VPN Server

Sebelum membahas apa itu *VPN Server*, perlu diperhatikan bagaimana arsitektur VPN ini bekerja. VPN atau lebih jelasnya *Virtual Private Network* dapat membantu mengamankan lalu lintas data privat melalui jaringan publik seperti Internet. VPN memberikan pengamanan untuk enkripsi dan enkapsulasi lalu lintas jaringan privat dan berpindah melalui jaringan dibawahnya. Data di enkripsi untuk langkah pengamanan, dan paket yang dicuri pada jaringan publik tidak dapat di dekripsi tanpa kunci enkripsi yang tepat. Data pun di enkapsulasi atau dibungkus dengan *IP header* berisi informasi *routing*.

VPN membantu pengguna untuk bekerja dari rumah, dijalan, atau di lobi kantor untuk langsung terhubung secara aman ke *server* perusahaan menggunakan internet. Dari sisi pengguna akan terlihat bahwa, VPN adalah koneksi dari titik ke titik antara komputer pengguna den-

gan *server* perusahaan. Koneksi jaringan aslinya, internet, tidak penting bagi pengguna karena data terlihat seperti dikirimkan dari jalur privat.

Ada beberapa cara menggunakan VPN. Cara yang paling umum adalah *Remote Access VPN*, ketika pengguna melakukan akses jaringan privat (yang biasa dikenal dengan LAN) melalui internet menggunakan akses koneksi VPN. Kondisi lain yaitu kantor cabang melakukan koneksi VPN ke perusahaan baik dengan koneksi yang tetap maupun berdasarkan kebutuhan (on-demand site-to-site) yang biasa dikenal dengan *router-to-router VPN connection*.

Tiap cara diatas dapat diimplementasikan untuk memberikan sambungan atau koneksi melalui jaringan publik, seperti internet, atau melalui intranet. Koneksi VPN juga dapat diimplementasikan untuk menghasilkan komunikasi data yang aman antara perusahaan yang saling bekerja sama.

Remote access VPN sendiri terbentuk oleh akses *client* secara *remote*, dimana sebuah komputer yang melakukan koneksi ke jaringan private dari luar jaringan lokal. *VPN Server* memberikan akses ke sumber daya jaringan dimana *VPN Server* terhubung. Seluruh paket yang dikirim melalui koneksi VPN akan berpangkal di *VPN client*. *VPN Client* melakukan otentikasi dirinya ke *VPN Server* dan sebagai timbal baliknya, *VPN Serer* akan mengotentikasi dirinya ke *VPN Client*.

Sedangkan *site-to-site VPN* menghubungkan satu atau lebih jaringan privat. Sebagai contoh, *site-to-site VPN* memungkinkan sebuah organisasi memiliki koneksi dengan dengan jaringan yang terpisah, atau memiliki koneksi dengan organisasi lain melalui koneksi internet. Jalur koneksi VPN melalui internet secara logika beroperasi seperti *Wide Area*

Network (WAN).

VPN Server menyediakan rute koneksi ke jaringan dimana *VPN Server* terhubung. Pada *site-to-site VPN*, paket yang beredar pada koneksi VPN tidak berpangkal di *router*. *Router* penghubung (dalam hal ini *VPN Client*) mengotentikasi dirinya ke *router* penjawab (yaitu *VPN Server*), kemudian sebagai balasan, *VPN Server* atau *router* penjawab akan mengotentikasi dirinya ke *router* penghubung.

Dengan menggunakan koneksi VPN berbasis internet, sebuah organisasi dapat mengatasi jarak yang jauh dengan mengambil kelebihan dari ketersediaan internet secara global.

Untuk *Remote access VPN* melalui internet memungkinkan *remote access client* memulai dengan koneksi *dial-up* ke ISP lokal, bukan langsung ke perusahaan. Dengan menggunakan koneksi fisik ke ISP lokal, *remote access client* memulai koneksi VPN melalui internet ke *VPN Server* perusahaan. Ketika koneksi VPN terbentuk, *remote access client* dapat mengakses sumber daya pada intranet perusahaan.

Untuk *site-to-site VPN* melalui internet, ketika jaringan sudah terkoneksi dengan internet, *router* melangsungkan paket ke *router* lain melalui koneksi VPN. Bagi *router*, koneksi VPN beroperasi seperti *data-link layer*.

Istilah *VPN Tunneling* adalah teknologi jaringan yang memungkinkan enkapsulasi dari satu tipe paket protokol ke dalam *datagram* dengan protokol yang berbeda. Sebagai contoh, koneksi VPN Windows dapat menggunakan paket *Point-to-Point Tunneling Protocol* (PPTP) untuk enkapsulasi dan mengirimkan melalui jaringan privat, seperti TCP/IP melalui jaringan publik seperti internet.

Bagi PPTP dan *Layer Two Tunneling Protocol* (L2TP), *tunnel* mirip

seperti *session*. Masing-masing ujung *tunnel* harus sepakat atas *tunnel* dan harus menegosiasikan variabel konfigurasi, seperti pemberian alamat, enkripsi, atau kompresi. Kebanyakan kasus, data dikirim melalui *tunnel* dengan menggunakan protokol berbasis *datagram*. Manajemen protokol *tunnel* digunakan untuk membuat, memelihara, dan mengakhiri *tunnel*.

Setelah *tunnel* terbentuk, data dapat dikirim. Baik *client* atau *server* tunnel menggunakan protokol transfer data *tunnel* untuk menyiapkan data yang akan dikirim. Ada 2 (dua) tipe pada *tunneling* ini, yaitu :

- *Voluntary Tunneling*

  Pengguna atau komputer *client* dapat meminta layanan VPN untuk melakukan konfigurasi dan membuat *voluntary tunnel*. Dalam hal ini pengguna atau komputer *client* adalah *tunnel endpoint* dan bertindak sebagai *tunnel client*.

  Voluntary Tunneling terjadi ketika komputer *client* atau *routing server* membuat koneksi virtual dengan *server tunnel*. Untuk menyempurnakan agar koneksi ini terjadi, perangkat lunak *tunneling client* dan protokol *tunneling* yang tepat harus terpasang pada komputer *client*.

  Pada kondisi *dial-up*, *client* perlu melakukan *dial-up* terlebih dahulu ke jaringan sebelum *client* dapat membangun *tunnel*. Ini kasus yang sering terjadi. Contoh yang bagus dari kasus *dial-up* ini adalah pengguna internet *dial-up*, yang mengharuskan pengguna melakukan *dial* ke ISP untuk mendapatkan koneksi internet sebelum *tunnel* yang melalui internet dapat dibangun.

  Untuk *client* yang terhubung dalam jaringan LAN, sudah ada koneksi jaringan yang memberikan *routing* enkapsulasi *payload* to

server *tunnel* LAN. Ini adalah salah satu contoh dimana *client* menggunakan koneksi *always-on broadband internet*.

Ada kekeliruan bahwa koneksi VPN membutuhkan koneksi *dial-up*. Yang sebetulnya dibutuhkan oleh koneksi VPN adalah alamat IP antara *VPN Client* dan *VPN Server*. Beberapa *client* (seperti komputer rumahan) menggunakan *dial-up* untuk mendapatkan koneksi internet untuk membangun transport IP. Ini adalah langkah pendahuluan pada tahap persiapan pembuatan *tunnel* dan bukan merupakan bagian dari protokol *tunnel* itu sendiri.

- *Compulsory Tunneling*

  Pada *compulsory tunneling*, VPN *remote access server* melakukan konfigurasi dan membuat *compulsory tunnel*. Pada *compulsory tunnel*, komputer pengguna bukan *tunnel endpoint*. Perangkat lain seperti *server dial-up*, yang berada diantara komputer pengguna dan *server tunnel* adalah *tunnel endpoint* dan beraksi sebagai *tunnel client*.

Point-to-Point Tunneling Protocol (PPTP) akan mengenkapsulasi frame *Point-to-Point Protocol* (PPP) kedalam IP *datagram* untuk nantinya dikirimkan melalui jaringan berbasis IP, seperti internet atau melalui intranet privat. PPTP menggunakan koneksi TCP. PPTP sebetulnya adalah hasil modifikasi dari *Generic Routing Encapsulation* (GRE) untuk mengenkapsulasi frame PPP menjadi data *tunnel*. Data dari frame PPP yang dienkapsulasi dapat di enkripsi, kompresi, atau dilakukan keduanya secara bersamaan.

Tunnel endpoint harus terotentikasi sebelum koneksi *VPN Tunnel* yang aman terjadi. Pengguna yang membuat akses VPN dapat menggunakan *password*, *biometric*, otentikasi dua arah, atau metode kriptografi yang

lain. Pada *tunnel network-to-network* biasanya menggunakan *password* atau sertifikat digital. *Tunnel* ini akan menyimpan kunci secara permanen agar *tunnel* dapat terhubung secara otomatis tanpa bergantung kepada administrator.

VPN tidak dapat membuat koneksi *online* melalui jaringan publik sepenuhnya anonim, tetapi biasanya meningkatkan kerahasiaan dan keamanan. Untuk menjaga informasi rahasia yang bersifat pribadi, VPN hanya memperbolehkan akses *remote* yang terotentikasi menggunakan protokol *tunneling* dan teknik enkripsi.

Beberapa protokol VPN yang aman adalah sebagai berikut :

- *Internet Protocol Security* (IPsec), yang awalnya dibangun oleh *Internet Engineering Task Force* (IETF) untuk IPv6, dimana IPSec ini wajib diimplementasikan sebagai standar dari IPv6 dengan RFC 6434 dimana sebelumnya hanya direkomendasikan. Standar ini pun digunakan secara luas pada IPv4 dan L2TP. Desain dari IPSec ini memenuhi banyak kebutuhan seperti otentikasi, integritas, dan konfidensial. IPSec menggunakan enkripsi, mengenkapsulasi paket IP didalam paket IPSec. Penguraian enkapsulasi terjadi pada titik akhir dari *tunnel*, dimana paket IP yang asli di dekripsi dan diteruskan ke tujuannya.
- *Transport Layer Security* (SSL/TLS) dan membuat *tunnel* seluruh lalu lintas jaringan (seperti yang dilakukan pada proyek OpenVPN) atau mengamankan koneksi perorangan. SSL VPN dapat membuat koneksi dari lokasi dimana IPSec tidak dapat menjangkaunya melalui *Network Address Translation* dan aturan *firewall*.
- *Datagram Transport Layer Security* (DTLS) yang biasanya digunakan pada Cisco AnyConnect VPN dan OpenConnect VPN untuk

memberikan solusi atas permasalahan SSL/TLS ketika *tunneling* dengan protokol UDP.

- *Microsoft Point-to-Point Encryption* (MPPE) bekerja dengan *Point-to-Point Tunneling Protocol* (PPTP).
- *Secure Shell* (SSH) VPN, OpenSSH menyediakan *VPN tunneling* untuk mengamankan koneksi *remote* ke jaringan baik publik maupun lokal. *Server* OpenSSH menyediakan *tunnel* yang aktif secara bersamaan dalam jumlah yang terbatas.

3. **Akses Internet**

Hal lain yang mendukung sistem layanan ini dapat bekerja adalah akses internet dengan kemampuan yang mencukupi. Artinya, lebar pita *bandwidth* harus mencukupi untuk transaksi menerima dan memberikan respon.

Sebagai contoh simulasi kasar dalam penggunaan *bandwidth* akses internet ini adalah sebagai berikut, dari sejarah data yang terekam sejak pengalihan PBB-P2 ke Daerah, transaksi terbanyak dalam periode harian adalah 21.165 transaksi, yaitu ditanggal 29 Agustus 2014, bila dalam sekali proses pencatatan transaksi pembayaran PBB-P2, data yang dikirim oleh Bank sebagai tempat pembayaran hanya berupa Nomor Objek Pajak dan Tahun Pajak yang diinginkan, Nomor Objek Pajak berjumlah 18 karakter, dan tahun pajak berjumlah 4 karakter, dengan total 22 karakter, jika 1 karakter = 1 byte, maka dibutuhkan 22 byte untuk sekali mengunduh atau menerima data transaksi, untuk tanggal kritis seperti di tanggal 29 Agustus 2014 kebutuhan akan unduh atau menerima data permintaan pencatatan pembayaran data transaksi hanya sebesar 455 kB (*kilo bytes*). Namun berbeda untuk kasus respon atau mengunggah data hasil pencatatan transaksi pembayaran yang sukses, bila data yang dikirim dari server DPPK ke Server Bank sebagai

tempat pembayaran memiliki struktur berikut :

- Nomor Objek Pajak = 18 karakter
- Tahun Pajak = 4 karakter
- Nomor Transaksi Penerimaan Daerah = 16 karakter
- Jumlah Pembayaran Pokok = 12 karakter
- Jumlah Pembayaran Denda = 12 karakter
- Nama Wajib Pajak = 30 karakter
- Alamat Objek Pajak = 45 karakter

Maka untuk menjawab atau merespon sebuah permintaan transaksi pembayaran akan membutuhkan lebar pita sebanyak 137 karakter atau 137 *byte*. Dengan perhitungan terburuk, dalam rangka menjaga ketersediaan layanan dalam merespon suatu permintaan, maka bahwa transaksi pada tanggal 29 Agustus 2014 terjadi pada waktu yang bersamaan, akan dibutuhkan lebar pita (*bandwidth*) sebanyak 2.899.605 *byte*, bila 1 kB adalah 1.024*byte*, dan 1 MB adalah 1.024 *kB*, maka dihasilkan kebutuhan lebar pita (*bandwidth*) sebesar 2,77 MB. Ini adalah lebar pita (*bandwidth*) minimal yang dibutuhkan, untuk menjaga apabila suatu saat kedepan akan ditemukan jumlah transaksi yang jauh lebih besar dari kejadian di tanggal 29 Agustus 2014.

Lebar pita atau *bandwidth* yang diperlukan ini adalah lebar pita atau *bandwidth* murni, karena di beberapa penyedia layanan, ada beberapa paket yang lebar pitanya atau *bandwidth* diatur dalam model berbagi (*sharing bandwidth*), yang biasanya ada pada paket internet untuk rumah tangga, dimana *bandwidth* atau lebar pita yang semisal ditawarkan 10 MB/s, maka pada kenyataan kecil kemungkinan mencapai 10 MB/s, karena *bandwidth*

yang ditetapkan oleh penyedia layanan, misalkan 100 MB/s akan dibagi kesesama pengguna paket 10 MB/s, yang pada akhirnya, apabila ada lebih dari 10 pelanggan menggunakan paket 10 MB/s atau kita tetapkan secara kasar bahwa ada 100 pelanggan menggunakan paket 10 MB/s, maka apabila dalam waktu bersamaan setiap pengguna menggunakan internet, rata-rata setiap pelanggan hanya akan mendapatkan *bandwidth* atau lebar pita sebesar 100 pelanggan dibagi 100 MB/s, atau hanya sebesar 1 MB/s saja. Ini berbeda dengan layanan yang menyediakan *bandwidth* secara independen untuk tiap pelanggan, atau dalam istilah yang kita kenal adalah paket internet *dedicated* yang seharusnya, *bandwidth* atau lebar pita yang diberikan murni sesuai jumlah pada saat ditawarkan.

Selain lebar pita (*bandwidth*) seperti dijelaskan diatas, untuk menjaga ketersediaan sambungan / koneksi, maka diperlukan 2 (dua) atau lebih sambungan atau koneksi internet dengan penyedia jasa yang berbeda. Hal ini untuk menjaga apabila salah satu koneksi internet mengalami kegagalan sambungan atau ada permasalahan lain sehingga koneksi gagal terjalin, maka sambungan atau koneksi dari penyedia jasa yang lain yang masih memiliki koneksi akan menggantikan posisi jaringan yang sedang bermasalah. Sehingga dalam fungsinya sebagai penyedia layanan *web services* dapat memberikan layanan yang maksimal selama 24 jam sehari, sepanjang 7 hari dalam seminggu.

Skenario lain yang dapat digunakan untuk menjaga ketersediaan sambungan atau koneksi atau dalam istilah kerennya adalah *availability* yaitu dengan membangun *server* lain yang berbeda lokasi secara geografis. Tentunya membangun sebuah *server* di luar DPPK Kabupaten Brebes akan membutuhkan biaya yang tidak sedikit. Namun dengan perkembangan teknologi informasi sekarang ini, layanan yang mungkin digunakan secara lebih efektif dan efisien

adalah menggunakan layanan *cloud* atau komputer awan. Layanan *cloud* ini sebetulnya berbentuk *server* virtual yang secara fisik terdiri dari beberapa server yang digabung menjadi sebuah layanan *cloud*, sebuah *server* virtual di *cloud* akan memanfaatkan *resources* atau sumber daya yang optimal dari *server-server* fisik yang menyusunnya.

Dengan menggunakan layanan *cloud* ini, maka bukan hanya melakukan penjagaan terhadap koneksi internet, tetapi juga melakukan duplikasi data sehingga apabila ada kerusakan data di *server* basis data DPPK maka data tetap utuh berada di server basis data *cloud*. Secara kasar mungkin akan terlihat mirip dengan sistem *single host* yang sekarang digunakan, yaitu ada *server* basis data di DPPK, dan ada *server* basis data di Bank tempat pembayaran, yang membedakan adalah pada sistem *single host*, struktur basis data di DPPK dan struktur basis data di Bank tempat pembayaran berbeda sangat jauh, sedangkan pada layanan *cloud* antara struktur basis data di DPPK dan struktur basis data di *server cloud* sama persis sehingga dapat dilakukan teknik *mirroring* atau penyamaan data antara basis data di DPPK dan layanan *server cloud*.

3 SUMBER DAYA MANUSIA YANG TERLIBAT DALAM SISTEM APLIKASI

Sumber daya yang terlibat dalam sistem aplikasi *web services* pencatatan pembayaran PBB-P2 ini secara langsung tidak ada, karena komunikasi yang terjadi antara *server* penyedia layanan *web services* yang berada di DPPK, dengan *client* yang berada di Bank sebagai tempat pembayaran terjadi secara sistem aplikasi dengan sistem aplikasi. Adapun sumber daya manusia yang secara tidak langsung terlibat adalah sumber daya manusia yang berada di tempat pembayaran,

yang melakukan pencatatan transaksi pembayaran secara aplikasi yang berbeda yang dibangun oleh Bank dengan sistem aplikasi yang akan dibangun yang berupa layanan *web services*.

4 ORGANISASI SISTEM APLIKASI

Organisasi dari sistem aplikasi ini boleh dikatakan cukup sederhana, karena hanya terdiri dari 3 (tiga) cara komunikasi antara *client* dan *server* dalam melakukan pencatatan transaksi pembayaran oleh wajib pajak atau kuasanya.

Yang pertama adalah proses *inquiry*, yaitu proses dimana *client* melakukan *request* terhadap informasi sebuah objek dan subjek pajak yang biasanya digunakan untuk klarifikasi data antara Surat Pemberitahuan Pajak Terhutang (SPPT) yang diserahkan wajib pajak atau kuasanya, dengan data yang tersimpan dalam basis data SISMIOP, seharusnya kedua data ini sama, baik dari nomor objek pajaknya, nama wajib pajaknya, dan besarnya Pajak Bumi dan Bangunan yang terhutang.

Bila ditemukan adanya perbedaan di salah satu dari 3 (tiga) komponen tersebut, artinya telah terjadi perubahan di basis data SISMIOP yang disebabkan oleh permohonan pelayanan PBB-P2 di DPPK Kabupaten Brebes, artinya, seharusnya SPPT yang diserahkan ke Bank sebagai tempat pembayaran sudah tidak berlaku, dan akan diterbitkan SPPT terbaru yang menandakan bahwa proses pengajuan pelayanan PBB-P2 telah selesai dilakukan dan SPPT terbarulah yang berlaku sebagai pengganti SPPT yang lama yang telah dicabut.

Transaksi yang kedua adalah pencatatan transaksi pembayaran, modul ini apabila dilakukan *request* terhadapnya, akan langsung melakukan pencatatan pembayaran di tabel tertentu pada basis data SISMIOP dan kemudian memberikan respon berupa informasi data yang telah dilakukan pencatatan pembayaran terhadapnya.

Transaksi yang ketiga adalah modul *reversal*, dimana terjadi pembatalan pembayaran yang dilakukan oleh Bank sebagai tempat pembayaran karena adanya kesalahan, baik kesalahan secara sistem maupun kesalahan SDM.

Hal-hal yang dimungkinkan terjadi karena kesalahan sistem adalah terputusnya koneksi antara *server* di DPPK dengan *client* di Bank tempat pembayaran. Pada saat *client* mengirimkan sebuah *request* untuk melakukan pencatatan pembayaran, karena koneksi terputus sehingga *server* yang mengirimkan respon data tidak diterima dengan baik oleh *client*, maka *client* seharusnya melakukan *request reversal* terhadap proses ini. Hal lain yang terjadi karena kesalahan sistem adalah apabila tiba-tiba *server* mengalami kerusakan, sedangkan *client* ingin melakukan transaksi pencatatan pembayaran. Dan banyak hal lain

Karena budaya pembayaran PBB-P2 di Kabupaten Brebes dimana wajib pajak melakukan pembayaran langsung kepada petugas pemungut di tingkat Desa/Kelurahan, maka hal yang sudah biasa dilakukan adalah petugas pemungut tingkat Desa/Kelurahan akan melakukan penyetoran pajak yang dilakukan dalam periode tertentu seperti mingguan atau bulanan, akibat yang ditimbulkan dari pola seperti ini adalah Bank sebagai tempat pembayaran akan menerima pembayaran dalam bentuk massal yang terkadang lebih dari 100 Objek Pajak atau transaksi pada hari yang sama.

Dari hal tersebut diatas, maka dibutuhkan proses yang cepat untuk melakukan pencatatan pembayaran, maka dibutuhkan strategi dalam komunikasi antara aplikasi dengan basis data agar prosesnya berjalan optimal. Hal yang mungkin untuk optimalisasi hal ini adalah dengan *stored procedure*.

Stored procedure adalah sebuah prosedur layaknya sub-program di dalam bahasa pemrograman reguler yang tersimpan di dalam katalog basis data. Karena berada dalam katalog basis data Oracle, jadi sebetulnya *stored procedure* ini berisi sekumpulan perintah SQL yang memiliki nama dan kegunaan tertentu.

Beberapa keuntungan dalam penggunaan *stored procedure* ini yaitu :

- Meningkatkan performa, karena *stored procedure* biasanya menggunakan bahasa *native* dari basis data yaitu SQL (*Structured Query Language*), sehingga tugas untuk menerjemahkan atau memproses data sudah selayaknya berada pada sistem basis data sehingga sistem aplikasi tidak terbebani dalam pengolahan basis datanya.

- Mereduksi lalu lintas data pada jaringan. Karena pemanggilan *stored procedure* hanya menggunakan beberapa karakter saja untuk membentuk nama prosedur beserta parameternya, maka tentu saja ini jauh lebih kecil dalam menggunakan *bandwidth* jaringan dibandingkan dengan penggunaan SQL murni yang melakukan perintah *query* dari beberapa tabel sekaligus.

- *Reusable*. Artinya dapat digunakan oleh aplikasi apapun tanpa harus membuatnya ulang. Setiap aplikasi yang akan melakukan olah data yang sama pada basis data tidak perlu menyusun *query* dari awal, melainkan cukup menggunakan *stored procedure* yang sudah ada dalam katalog basis data. Begitu pula apabila ada perubahan aturan bisnis, hal yang perlu diubah apabila menyangkut manipulasi data pada basis data, maka cukup

- Meningkatkan kontrol keamanan. Dalam kondisi tanpa penggunaan *stored procedure*, *query* akan dikirimkan dari aplikasi ke sistem basis data dalam bentuk teks mentah, sehingga parameternya dapat dimanipulasi dan menyebabkan kerusakan basis data yang cukup parah. Hal ini biasa dikenal dengan serangan *SQL Injection*.

  Sebuah contoh dari serangkat *SQL Injection* ini adalah sebagai berikut :

  Apabila menggunakan *query* (SQL) biasa terhadap basis data, maka langkah yang diambil untuk melakukan pengecekan data *password* dalam basis data

akan menggunakan perintah berikut :

```
1     select * from user where username = :username
2
```

Yang artinya, mengambil seluruh informasi tentang *user* entah itu nama pengguna, *password*, tanggal lahir, atau seluruh informasi yang berkaitan dengan *user* dengan filter bahwa *username* harus sama dengan apa yang dimasukkan oleh pengguna ke variabel `:username`. Kondisi akses ini akan diterima dan *user* diijinkan masuk ke dalam aplikasi apabila ada *user* dengan *username* yang tersimpan pada basis data. Namun seseorang dapat mengisikan variabel `:username` seperti ini :

```
1     ''; select * from user
2
```

Tanda petik tunggak 2 (dua) kali artinya memberikan parameter kosong ke variabel yang dibutuhkan, sedangkan kode tanda titik koma di awal baris menyatakan bahwa ada perintah sebelumnya yang harus dijalankan, kemudian jalankan perintah berikutnya, dimana dalam konteks ini adalah `select * from user`. Yang artinya mengambil seluruh data *user* tanpa filter sama sekali, ambil seluruh data *user* yang berada pada basis data. Kemudian kode SQL akan dieksekusi seperti ini ;

```
1     select * from user where username = ''; select * from user
2
```

Yang artinya, apapun hasil dari *query* awal, aplikasi akan secara otomatis dapat diakses tanpa melakukan proses *login* yang benar, karena data *user* dan *password* nya dapat diakses tanpa menggunakan filter dengan *query* yang kedua.

Akan lebih berbahaya apabila ada pengguna yang menambahkan atau

mengisikan variabel `:username` dengan perintah berikut :

```
1    ''; drop database {nama database}
2
```

Akibatnya seluruh data yang tersimpan pada basis data akan hilang tanpa diketahui apa penyebabnya.

5 WAKTU DAN BIAYA YANG DIBUTUHKAN DALAM PEMBUATAN/PENGEMBANGAN SISTEM APLIKASI SECARA MENYELURUH

Karena keterbatasan sumber daya manusia, dalam hal ini pemrogram *programmer*, yaitu hanya 1 (satu) orang, maka waktu yang dibutuhkan untuk membangun sistem aplikasi secara menyeluruh akan diperkirakan selama 3 (tiga) bulan dan dikerjakan pada jam kerja. Kurang lebih waktu yang dibutuhkan akan melihat dari kendala-kendala yang ditemui pada saat operasional dilakukan pengembangan sistem aplikasi ini. Namun perkembangannya dapat dilihat secara mingguan, bahkan harian pada *server repository* Github.

Github sendiri sebetulnya memiliki basis Git, Git sendiri sebetulnya adalah *version control system*, yaitu sistem yang melakukan kontrol terhadap versi sistem aplikasi, yang mengatur dan menyimpan revisi terhadap sebuah proyek pengembangan sitem aplikasi.

Walaupun seringkali digunakan untuk menyimpan kode program, Git sebenarnya mampun untuk mengatur jenis berkas yang lain seperti dokumen *Word* ataupun proyek *Final Cut*.

Beberapa pendahulu Git seperti CVS dan Subversion memiliki *repository* terpusat dari seluruh berkas yang berkaitan dengan proyek. Ketika seorang pengembang melakukan perubahan terhadap kode, perubahan ini akan tercatat langsung di *repository* pusat. Namun dengan *version control system* terdistribusi seperti Git, jika ingin melakukan perubahan pada proyek, perlu menyalin seluruh proyek ke komputer yang digunakan terlebih dahulu. Lakukan perubahan pada hasil duplikasi di komputer lokal, kemudian kirimkan perubahannya ke *server* pusat. Ini menjadikan pengiriman perubahan ke *server* pusat menjadi potongan-potongan kecil sehingga tidak perlu melakukan koneksi dengan *server* setiap kali akan melakukan perubahan.

Github sendiri adalah layanan *hosting repository* untuk Git, tetapi Github menambahkan banyak sekali fitur. Jika Git adalah perkakas yang berbasis *command line*, seperti menggunakan DOS pada Windows atau Shell pada Linux, maka Github menyediakan tampilan grafis berbasis *web*. Github pun memberikan kontrol akses dan beberapa fitur kolaborasi, seperti *wiki* dan perkakas manajemen tugas dasar untuk setiap proyek.

Semboyan dari fungsionalitas Github sendiri adalah *"forking"*, yaitu menyalin atau membuat duplikasi *repository* dari pengguna yang satu ke pengguna yang lain. Ini memungkinkan pengambilan proyek yang seorang pengguna tidak memiliki akses untuk merubahnya, kemudian melakukan perubahan atau modifikasi proyek pada akunnya sendiri. Jika kemudian akan membagikan hasilnya, dapat dilakukan *"pull request"* ke pemilik asli proyek tersebut. Yang kemudian seorang pengguna dapat menekan sebuah tombol yang menggabungkan perubahan-perubahan yang terjadi ke proyek asalnya.

Ketiga fitur ini (yaitu *fork*, *pull request*, dan *merge*) menjadikan Github tangguh. Sebelum adanya Github, jika ingin melakukan kontribusi ke proyek *open source*, perlu melakukan unduh penuh kode sumber proyek, kemudian melakukan

perubahan atau modifikasi pada komputer lokal, membuat daftar perubahan yang biasa dikenal dengan *"patch"*, kemudian mengirimkan *email* yang berisi daftar *patch* ke seseorang atau tim yang menangani pemeliharaan proyek. Kemudian orang-orang yang menangani pemeliharaan proyek ini melakukan evaluasi terlebih dahulu terhadap *patch* yang diterima, kemudian memutuskan apakah akan menggabungkan perubahannya kedalam sistem aplikasi yang ada atau tidak.

Disinilah efek jaringan berperan pada Github. Ketika seseorang melakukan *pull request*, orang-orang yang bertugas melakukan pemeliharaan proyek dapat melihat profil dari orang yang melakukan *pull request*, termasuk apa saja yang telah orang ini kontribusikan di Github. Jika *patch* diterima, orang yang melakukan *pull request* mendapatkan penghargaan di situs asli proyek tersebut, kondisi ini pun akan muncul di profil orang yang melakukan kontribusi. Ini seperti ikhtisar yang membantu orang-orang yang melakukan pemeliharaan proyek untuk menentukan bagaimana reputasi kontributor tersebut. Semakin banyak orang dan proyek di Github, semakin baik gambaran ide yang para pemelihara proyek akan mendatangkan kontributor yang potensial. *Patch* juga dapat didiskusikan secara publik di Github.

Bahkan bagi orang-orang yang melakukan pemeliharaan yang tidak menggunakan *interface* Github, Github dapat membuat manajemen kontribusi semakin mudah, cukup mengunduh *patch*, kemudian melakukan *merge* pada *command line*. Tetapi kelebihan lain dengan menggunakan *interface* Github adalah Github menyediakan tempat terpusat bagi para kontributor melakukan diskusi terhadap *patch* yang masuk.

Selain menawarkan *repository* publik, Github juga menyediakan *repository* privat bagi skala *enterprise* dimana biasanya kode adalah hak intelektual perusahaan. Solusi ini tentu saja tidak dapat mengambil sisi positif dari efek jaringan Github, tetapi dapat memanfaatkan sisi positif dari fitur kolaborasi. Beginilah cara Github

mencari sumber dana, dan Github tidak sendirian di pangsa pasar seperti ini.

Sumber penghasilan Github mungkin dari fasilitas *repository* private ini, tetapi jiwanya ada pada *repository* publik. Yang lebih penting adalah bahwa Github telah menjadi perpustakaan Alexandria bagi contoh kode. Karena Git mencatat perubahan-perubahan, *programmer*, entah itu pemula atau yang sudah mahir, dapat melihat langkah-langkah apa saja yang telah diambil oleh *developer* terbaik di dunia dan mencari tahu bagaimana mereka mengambil solusi atas permasalahan-permasalahan yang muncul.

Dari sisi biaya yang dibutuhkan, maka pengembangan sistem aplikasi ini sebetulnya tidak membutuhkan biaya sama sekali, hanya saja faktor pendukung sebagai alat komunikasi ke Bank sebagai tempat pembayaran perlu dimatangkan kembali, mengingat koneksi internet yang sekarang terpasang memiliki spesifikasi *shared bandwidth* dimana *bandwidth* yang diberitakan dalam promosi penyedia layanan internet, tidak sepenuhnya didapatkan melainkan dibagi berdasarkan jumlah pengguna yang mengambil paket yang sama. Ini berbeda dengan paket atau layanan *dedicated* yang ditawarkan oleh penyedia jasa layanan internet, yang biasanya layanan *dedicated* memiliki kapasitas *bandwidth* yang murni seperti pada promosi yang ditawarkan. Dan biasanya akan diberikan alamat IP Publik yang dapat dimanfaatkan sebagai identitas alamat jaringan DPPK Kabupaten Brebes.

Layanan *dedicated* ini pun jauh lebih mahal daripada layanan biasa, karena memang pelayanan dan kualitas yang ditawarkan berbeda.

Kebutuhan lain akan peralatan komunikasi ini adalah *Router* dan *VPN Server* yang biasanya disediakan oleh satu alat yang tergabung. Router sendiri berfungsi sebagai penghubung komunikasi antara 2 (dua) jaringan secara logika dan fisik, karena alamat IP jaringan dari misalkan 192.168.2.0 dengan *netmask* 255.255.255.0 akan berbeda alamat IP jaringan 192.168.1.0 dengan *netmask* 255.255.255.0. Disinilah fungsi *router*, yaitu untuk menghubungkan kedua jaringan seperti ini

agar dapat saling berkomunikasi dan saling mengirimkan data.

VPN Server sendiri adalah *server* yang melakukan manajemen terhadap jaringan virtual yang bersifat privat. Karena kondisi komunikasi pencatatan pembayaran PBB-P2 antara DPPK dan Bank sebagai tempat pembayaran dilakukan melalui jalur internet, maka diperlukan sebuah sistem agar data yang dikirimkan aman.

6 MANFAAT DAN DAMPAK DARI PENGEMBANGAN SISTEM APLIKASI

Manfaat dan dampak yang ditimbulkan dari pengembangan sistem aplikasi ini adalah terbentuknya data yang konsisten sehingga data dapat dioleh baik secara periodik maupun *realtime*, tidak seperti sekarang yang harus ada proses sinkronisasi data antara DPPK dengan Bank sebagai tempat pembayaran.

Manfaat lain yang didapat adalah karena data transaksi pembayaran dicatatkan langsung dalam basis data SISMIOP, maka realisasi dapat disajikan secara *realtime*, termasuk perubahan-perubahan yang terjadi dalam rangka pelayanan PBB-P2.

Manfaat lain adalah dapat terbukanya kerjasama dengan Bank yang lain sebagai tempat pembayaran sehingga memperluas cakupan wilayah tempat pembayaran sehingga mempermudah wajib pajak melakukan pembayaran PBB-P2 terutama bagi wajib pajak yang berada di luar wilayah Kabupaten Brebes, atau Provinsi Jawa Tengah.