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

14 November 2016

Priyanto Tamami, S.Kom.

1 SISTEM KOMPUTER

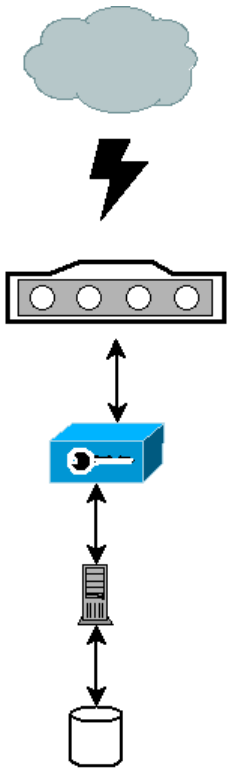
Sistem komputer yang digunakan akan terbagi menjadi 3 (tiga) yaitu :

1. Sebagai *server* basis data adalah sebagai berikut :
   - Prosesor Intel Xeon 2,4GHz
   - Memori 44GB
   - Sistem Operasi Windows Server 2008 R2 64 bit.
2. Sebagai *server* aplikasi adalah sebagai berikut :
   - Prosesor Intel Xeon 3,1GHz
   - Memori 4GB
   - Sistem Operasi CentOS 6.2 64 bit
3. Sebagai *client* spesifikasi yang digunakan bebas, dapat menggunakan sistem komputer apapun yang dapat berkomunikasi melalui jaringan TCP/IP.

Karena *client* nantinya adalah Bank sebagai tempat pembayaran, maka sebagai sarana untuk uji coba dapat menggunakan sistem komputer apapun dengan *browser* Chrome / Firefox.

2 SISTEM JARINGAN

Sistem jaringan yang nantinya dibangun akan terlihat seperti pada gambar 1 :



Gambar 1: Diagram Sistem Jaringan *Web Services* PBB

Dari diagram tersebut, gambar awan adalah simbol untuk jaringan internet. Untuk melakukan akses ke *server web service* akan melalui modem dibawahnya, kemudian akan menghubungi VPN *server* terlebih dahulu untuk mendapatkan otentikasi atau akses ke dalam jaringan internal.

Setelah sukses melakukan otentikasi ke *server* VPN, selanjutnya *client* dalam hal ini Bank akan melakukan akses ke *server web service* langsung, dimana *server web service* akan melakukan komunikasi dengan *server* basis data.

3 SISTEM BASIS DATA

Sistem basis data yang digunakan adalah sama dengan sistem basis data yang digunakan pada SISMIOP untuk pengelolaan PBB-P2, yaitu sistem basis data Oracle Database 11g. Namun tidak semua objek digunakan pada sistem *web service* yang akan dibangun. Beberapa tabel yang digunakan adalah sebagai berikut :

- Tabel SPPT

  Tabel SPPT ini nantinya hanya akan merubah pada kolom STATUS_PEMBAYARAN_SPPT, untuk isian 0 (nol) artinya nomor objek pajak (NOP) untuk tahun pajak tersebut belum dibayarkan, sedangkan isian 1 (satu) artinya NOP untuk tahun pajak tersebut sudah terbayar.

- Tabel PEMBAYARAN_SPPT

  Tabel PEMBAYARAN_SPPT ini apabila ada transaksi pembayaran, akan dicatatkan lengkap pada tabel ini, NOP apa, tahun pajak kapan, besarnya nilai yang dibayarkan, tanggal pembayaran, semuanya tersimpan pada tabel ini. Namun bila terjadi permintaan proses *reversal*, maka data yang tersimpan pada tabel ini akan dihapuskan.

- Tabel LOG_TRX_PEMBAYARAN

  Tabel LOG_TRX_PEMBAYARAN ini digunakan untuk menyimpan aktivitas transaksi pembayaran yang sukses.

- Tabel LOG_REVERSAL

  Tabel LOG_REVERSAL digunakan untuk menyimpan aktivitas *reversal* yang berhasil dilakukan pada basis data.

Beberapa *store procedure* juga dibuat untuk mempercepat eksekusi proses, *store procedure* ini, *store procedure* yang dibuat adalah sebagai berikut :

- SPPT_TERHUTANG

  Store procedure ini akan bertugas memberikan data SPPT terhutang ke aplikasi yang melakukan eksekusi terhadapnya.

- PROSES_PEMBAYARAN

  Store procedure ini akan bertugas melakukan pencatatan pembayaran pada tabel PEMBAYARAN_SPPT, melakukan perubahan isi kolom STATUS_PEMBAYARAN_SPPT pada tabel SPPT, kemudian melakukan pencatatan prosesnya pada tabel LOG_TRX_PEMBAYARAN.

- REVERSAL_PEMBAYARAN

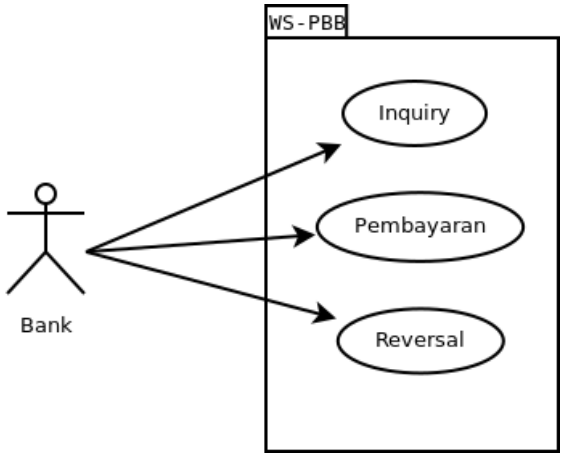
  Store procedure ini bertugas melakukan penghapusan data pada tabel PEMBAYARAN_SPPT, merubah kolom STATUS_PEMBAYARAN_SPPT pada tabel SPPT menjadi 0 (nol), dan melakukan pencatatan pada tabel LOG_REVERSAL.

4 PROSEDUR AKTIVITAS

Prosedur aktivitas yang berlaku untuk sistem *web services* ini akan terbagi menjadi beberapa skenario yang dituangkan dalam beberapa diagram untuk mempermudah pemahaman alur aktivitas yang terjadi. Berikut adalah diagram yang terbentuk.

4.1 Diagram *Use-Case*

Pembahasan skenario utama terdapat pada diagram *use-case* seperti pada gambar 2 :



Gambar 2: Diagram *Use-Case*

Seperti terlihat pada diagram, skenario utama atau fitur dari *web service* PBB-P2 akan melayani *inquiry*, transaksi pembayaran, dan *reversal* pembayaran.

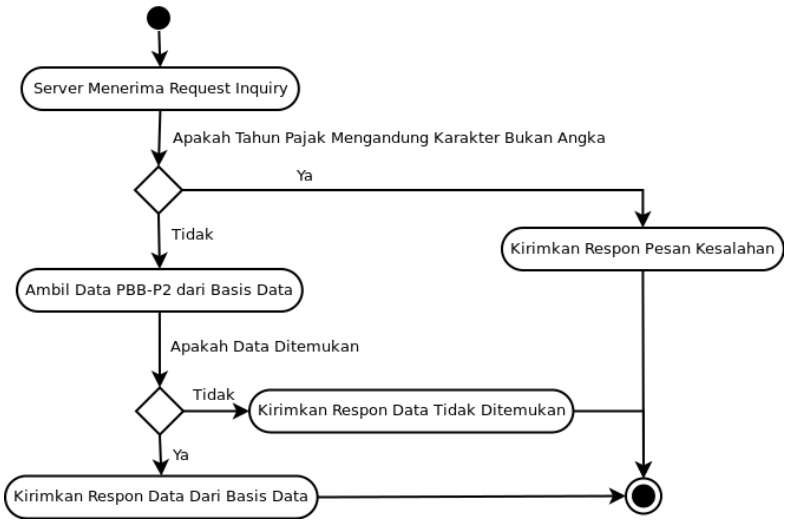
Secara mendalam, tiap skenario utama nantinya akan terbagi menjadi beberapa skenario rinci yang dijelaskan dalam beberapa diagram berbeda.

4.2 Diagram *Activity*

Dari 3 (tiga) skenario utama, akan muncul aktivitas detail yang digambarkan oleh diagram *activity* berikut :

4.2.1 Diagram *Activity Inquiry*

Pada diagram untuk skenario ini akan menjelaskan gambaran aktivitas *inquiry* yang terjadi dan bagaiamana respon terhadap *request inquiry* diselesaikan. Diagramnya seperti terlihat pada gambar 3 :



Gambar 3: Diagram *Activity Inquiry*

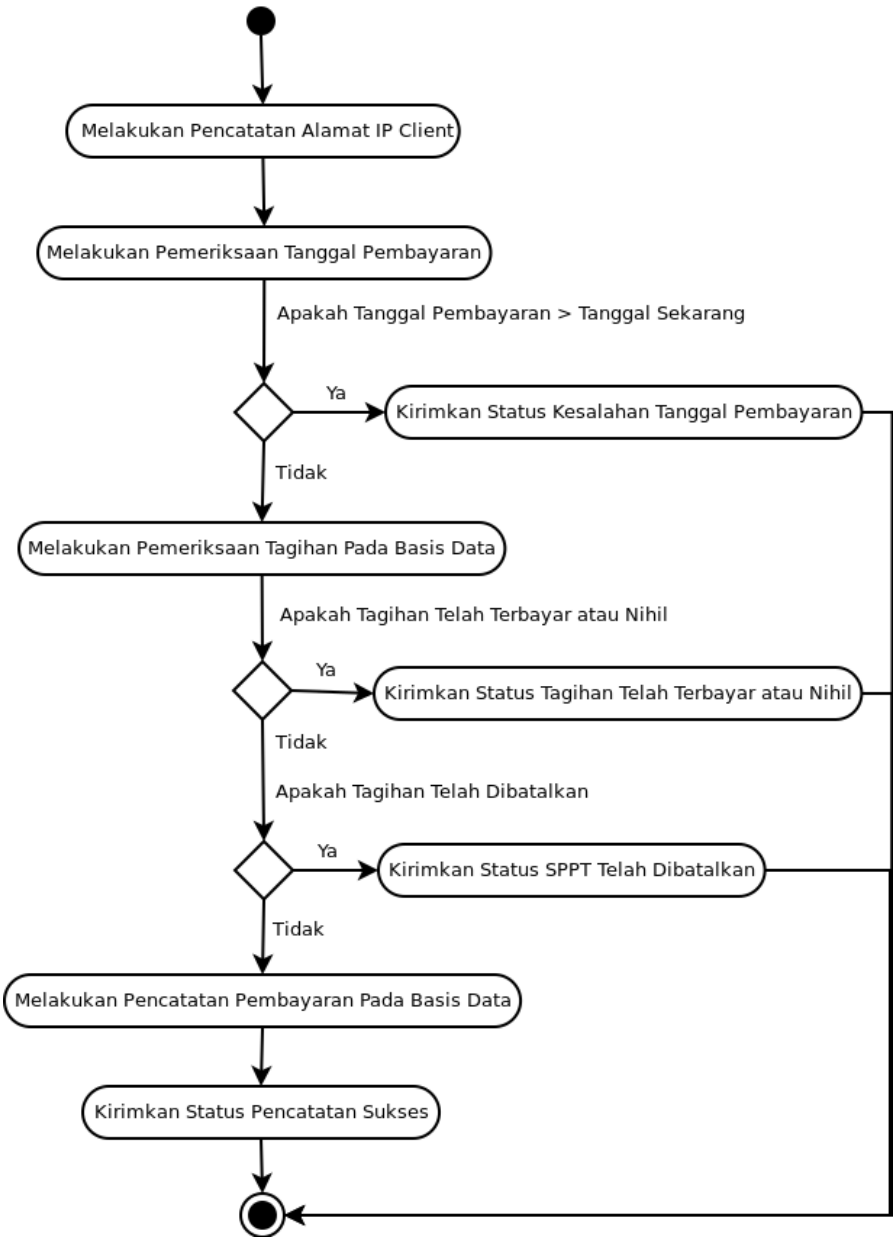
Skenario ini diawali dari *server* yang menerima *request inquiry* dari *client*, kemudian *server* melakukan pemeriksaan pada parameter tahun pajak yang diberikan, apakah mengandung karakter bukan angka atau merupakan karakter angka seluruhnya. Bila parameter tahun pajak mengandung karakter bukan angka, maka *server* akan mengirimkan respon pesan kesalahan ke *client*. Namun bila berhasil, maka melanjutkan ke aktivitas berikutnya yaitu mengambil data PBB-P2 pada basis data.

Hasil dari pengambilan data PBB-P2 pada basis data ada 2 (dua) kemungkinan, yaitu data ditemukan, dan data tidak ditemukan pada basis data.

Bila data tidak ditemukan pada basis data, maka *server* akan mengirimkan respon pesan bahwa data yang diminta tidak ditemukan dalam basis data, sedangkan bila data ditemukan, maka *server* akan memberikan respon pesan berupa data-data yang dibutuhkan dari basis data.

4.2.2 Diagram *Activity* Pencatatan Pembayaran

Diagram ini akan menjelaskan aktivitas yang terjadi pada saat *request* pencatatan pembayaran diterima oleh *server*, bentuk diagramnya akan terlihat seperti pada gambar 4 :



Gambar 4: Diagram *Activity* Pencatatan Pembayaran

Bila ada *request* pencatatan pembayaran, langkah yang pertama adalah melakukan pencatatan alamat IP *client*, kemudian melakukan pemeriksaan parameter tanggal pembayaran yang diberikan *client*, apakah tanggal pembayaran

terjadi setelah tanggal dan jam saat ini, atau sebelum tanggal dan jam saat ini.

Bila tanggal dan jam pembayaran melewati tanggal dan jam saat ini, maka *server* akan mengirimkan status kesalahan tanggal pembayaran dan proses selesai sampai disini.

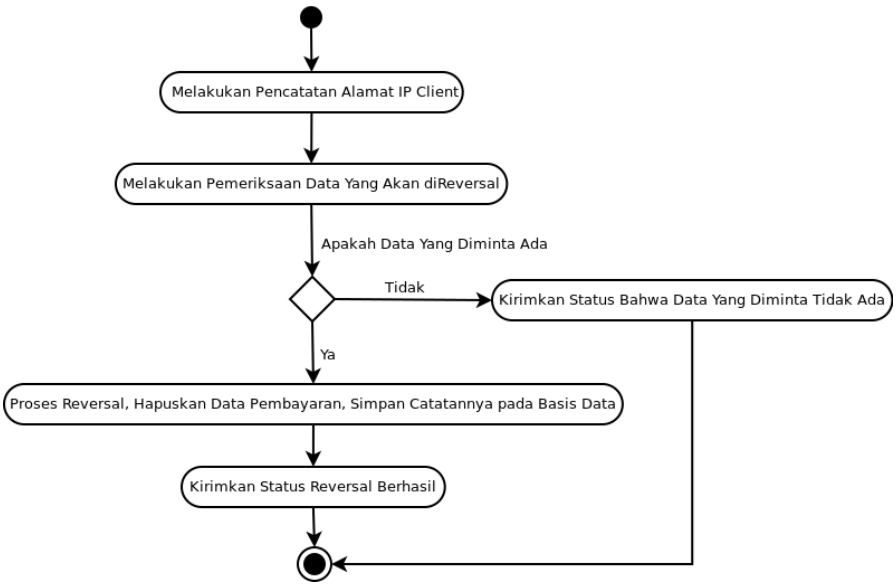
Ketika tanggal dan jam pembayaran sebelum tanggal dan jam saat ini, proses dilanjutkan dengan melakukan pemeriksaan tagihan pada basis data. Bila tagihan untuk data yang diminta berdasarkan Nomor Objek Pajak (NOP) dan tahun pajak telah terbayar atau nihil pada basis data, maka *server* akan mengirimkan status bahwa tagihan tersebut telah terbayar atau tagihan nihil kepada *client* dan proses selesai sampai disini.

Pemeriksaan pun dilanjutkan apakah NOP untuk tahun pajak tersebut telah dibatalkan atau tidak. Bila telah dibatalkan maka *server* akan mengirimkan status bahwa tagihan atas NOP untuk tahun pajak tersebut telah dibatalkan.

Bila seleksi atau pemeriksaan atas NOP untuk tahun pajak yang diminta muncul tagihan dan belum terbayar, status tagihannya pun tidak dibatalkan, maka *server* melakukan pencatatan pembayaran pada basis data, dan mengirimkan status sukses ke *client* sebagai sinyal bahwa *request* pencatatan pembayaran telah berhasil dilakukan. Sampai sini proses pencatatan pembayaran selesai.

4.2.3 Diagram *Activity Reversal*

Diagram ini akan menjelaskan aktivitas yang terjadi apabila ternyata ada kesalahan pencatatan pembayaran, sehingga aktivitas pencatatan pembayaran yang telah terjadi harus dikembalikan pada kondisi semula. Diagram aktivitas ini adalah seperti pada gambar 5 :



Gambar 5: Diagram *Activity* Untuk *Reversal* Pencatatan Pembayaran

Setiap ada *request reversal* yang diterima, maka *server* akan melakukan pencatatan alamat IP *client*, kemudian akan dilakukan pemeriksaan data yang akan dilakukan *reversal*, apakah data yang diinginkan ada pada basis data atau tidak.

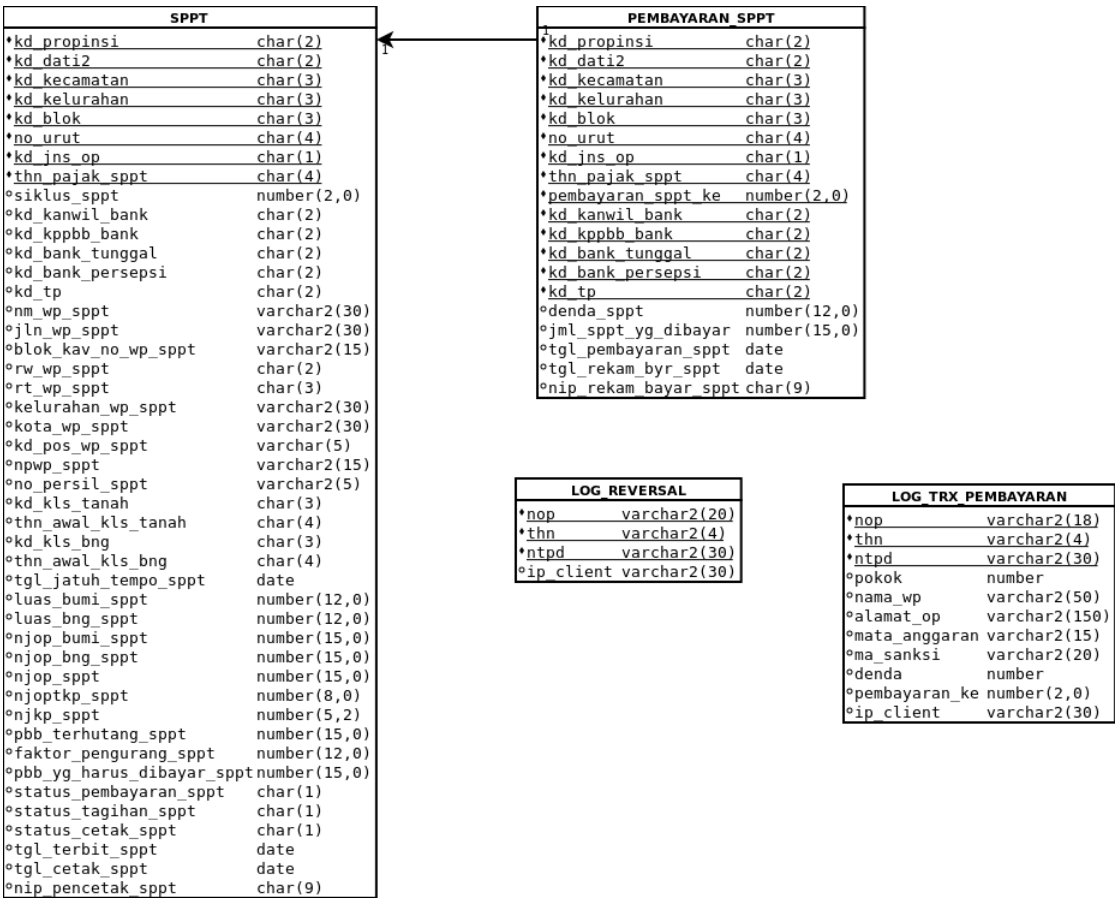
Bila data tidak ditemukan pada basis data, maka *server* akan mengirimkan status ke *client* bahwa data yang diminta untuk dilakukan *reversal* tidak ada pada basis data.

Bila data ditemukan pada basis data, maka *server* akan melakukan proses *reversal* dengan cara menghapus data pembayaran, dan melakukan pencatatan aktivitas *reversal* pada tabel terpisah, terakhir adalah mengirimkan status ke *client* bahwa proses *reversal* yang diminta telah berhasil dilakukan.

4.3 Diagram *Entity Relational*

Diagram ini akan memperlihatkan struktur tabel dan hubungan antar tabel yang digunakan pada sistem aplikasi *web services* PBB-P2. Diagramnya seperti terlihat

pada gambar 6 :



Gambar 6: Diagram *Entity Relational*

Pada tabel SPPT akan berisi seluruh tagihan PBB-P2 untuk seluruh tahun pajak. Tabel ini akan menjadi induk atau acuan dari tabel-tabel yang nantinya digunakan pada sistem ini. Namun kondisi yang nantinya berubah karena interaksi sistem *web service* dengan basis data dari tabel SPPT ini hanya pada kolom STATUS_PEMBAYARAN_SPPT.

Tabel yang kedua adalah tabel PEMBAYARAN_SPPT, yang menjadi tempat tampungan data dari transaksi pembayaran. Seluruh penerimaan / pembayaran PBB-P2 yang terjadi harus tercatat pada tabel ini. Kolom KD_PROPINSI,

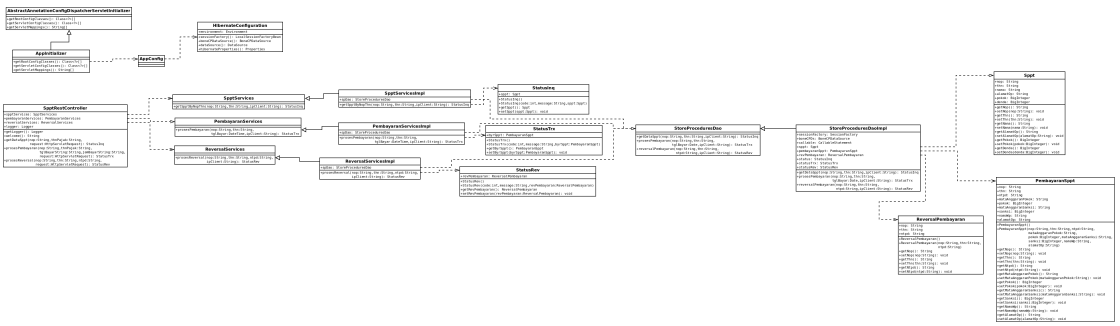
KD_DATI2, KD_KECAMATAN, KD_KELURAHAN, KD_BLOK, NO_URUT, KD_JNS_OP, dan THN_PAJAK_SPPT akan mengacu pada tabel SPPT.

Tabel ketiga adalah tabel LOG_TRX_PEMBAYARAN, tabel ini berfungsi untuk melakukan pencatatan aktivitas sistem, setelah melakukan perubahan pada kolom STATUS_PEMBAYARAN_SPPT milik tabel SPPT dari 0 (nol) menjadi 1 (satu), dan setelah melakukan pengisian data pada tabel PEMBAYARAN_SPPT, maka dicatatkan pula informasi aktivitasnya pada tabel LOG_TRX_PEMBAYARAN.

Tabel berikutnya adalah tabel LOG_REVERSAL, tabel ini berfungsi untuk mencatat aktivitas *reversal* yang terjadi, setelah kondisi kolom STATUS_PEMBAYARAN_SPPT pada tabel SPPT berubah dari 1 (satu) menjadi 0 (nol, dan setelah penghapusan data pada tabel PEMBAYARAN_SPPT.

4.4 Diagram *Class*

Diagram ini akan memperlihatkan kelas atau objek apa saja yang terbentuk untuk membangun sistem *web service* ini dapat bekerja sebagaimana mestinya, diagram *class* adalah sebagaimana gambar 7 :



Gambar 7: Diagram *Class*

kelas-kelas tersebut akan dijelaskan sebagai berikut :

1. *Interface* AbstractAnnotationConfigDispatcherServletInitializer

   Interface ini sebetulnya bawaan dari *framework* Spring yang digunakan untuk melakukan inisialisasi parameter awal dari sistem yang akan digunakan. Artinya setiap kelas yang melakukan *implement* terhadap *interface* ini nantinya akan digunakan sebagai pintu masuk awal sistem melakukan pengaturan-pengaturan / inisialisasi.

2. Kelas AppInitializer

   Kelas inilah yang mengimplementasikan *interface* AbstractAnnotationConfigDispatcherServletInitializer, sehingga dari kelas inilah sistem akan memulai aktivitasnya.

3. Kelas AppConfig

   Kelas ini digunakan sebagai tempat untuk konfigurasi sistem *web service* yang nantinya berjalan. Setelah sistem melakukan inisialisasi awal di kelas AppInitializer, kelas AppConfig ini nantinya melakukan konfigurasi kelas-kelas mana yang akan berhubungan dengan basis data, kelas mana yang nantinya melakukan konfigurasi hubungan antar kelas dan cara interaksinya.

4. Kelas HibernateConfiguration

   Kelas ini nantinya menjadi tempat konfigurasi *framework* Hibernate sebagai penghubung antara sistem aplikasi dengan basis data.

5. Kelas SpptRestController

   Kelas ini adalah tempatnya sistem aplikasi berhubungan dengan *client*. Setiap *request* yang dikirim oleh *client* akan masuk melalui kelas ini.

6. *Interface* SpptServices

Interface ini nantinya akan menyiapkan beberapa *method* yang akan diimplementasikan di kelas turunannya, sehingga kelas turunan dapat mengolah dan menyajikan berbagai data yang berhubungan dengan informasi SPPT. *Interface* ini pula yang nantinya menghubungkan kelas SpptRestController dengan beberapa *service* yang dibutuhkan dan berhubungan dengan SPPT.

7. *Interface* PembayaranServices

   Interface ini seperti *interface* SpptServices, yaitu menyiapkan *method* yang akan diimplementasikan di kelas turunan yang dapat mengolah dan menyajikan atau memproses data yang berhubungan dengan proses pencatatan pembayaran.

8. *Interface* ReversalServices

   Sebagaimana *interface* SpptServices dan PembayaranServices, *interface* ini pun hanya menyiapkan *method* yang akan diimplementasikan di kelas turunan yang berhubungan dengan kegiatan *reversal* pembayaran.

9. Kelas SpptServicesImpl

   Kelas ini adalah implementasi dari *interface* SpptServices yang akan mengolah dan menyajikan data terkait dengan *request inquiry* data SPPT.

10. Kelas PembayaranServicesImpl

    Kelas ini adalah implementasi dari *interface* PembayaranServices yang akan mengolah dan menyajikan data terkait dengan proses pencatatan pembayaran PBB-P2.

11. Kelas ReversalServicesImpl

    Kelas ini adalah implementasi dari *interface* ReversalServices yang akan mengolah dan menyajikan data terkait dengan proses *reversal* pembayaran PBB-P2.

12. Kelas StatusInq

    Kelas ini akan menampung informasi untuk proses *inquiry*, nantinya dari kelas ini akan secara otomatis menggunakan pustaka Jackson diubah ke format JSON sebagai bentuk komunikasi ke *client*.

13. Kelas StatusTrx

    Kelas ini akan menampung informasi yang dihasilkan dari aktivitas pencatatan transaksi pembayaran, kelas ini pun nantinya akan diubah otomatis oleh pustaka Jackson ke bentuk JSON sebagai format komunikasi dengan *client*.

14. Kelas StatusRev

    Kelas ini akan menampung informasi yang dihasilkan dari proses *reversal* pembayaran, kelas ini juga akan diubah secara otomatis ke bentuk JSON oleh pustaka Jackson sebagai format baku komunikasi dengan *client*.

15. *Interface* StoreProceduresDao

    Interface ini akan menyiapkan *method* yang digunakan oleh kelas turunan sebagai bahan untuk berkomunikasi dengan basis data. Tujuan lain dari pembentukan *interface* ini pun untuk mempermudah *framework* melakukan pemetaan mana kelas yang nantinya akan berhubungan dengan basis data.

16. Kelas StoreProceduresDaoImpl

    Kelas ini adalah implementasi dari *interface* StoreProceduresDao, nantinya setiap eksekusi *store procedure* milik basis data akan dilakukan dari kelas ini. Dari kelas ini pun hasilnya nanti didapat dan dikembalikan ke *client*.

17. Kelas Sppt

    Kelas ini nantinya akan menampung hasil dari pemanggilan *store procedure* milik basis data untuk aktivitas *inquiry* data PBB-P2.

18. Kelas PembayaranSppt

    Kelas ini akan menampung hasil dari pemanggilan *store procedure* milik basis data untuk proses pencatatan pembayaran PBB-P2.

19. Kelas ReversalPembayaran

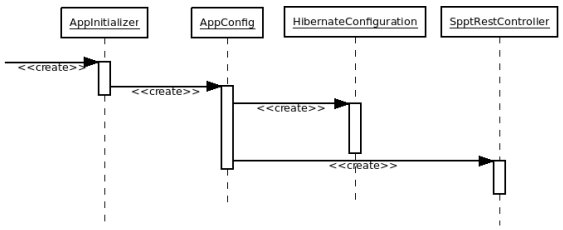
    Kelas ini akan menampung hasil dari pemanggilan *store procedure* milik basis data untuk proses *reversal* pembayaran PBB-P2.

4.5 Diagram *Sequence*

Diagram *sequence* akan menggambarkan interaksi antar kelas berdasarkan skenario-skenario yang telah disusun sehingga sistem aplikasi *web service* nantinya akan berjalan sebagaimana mestinya. Adapun skenario-skenario tersebut akan dibagi seperti berikut ini :

4.5.1 Skenario Konfigurasi *Spring Framework*

Pada skenario ini akan dijelaskan bagaimana tiap kelas berinteraksi pada saat sistem aplikasi memulai inisialisasi awal. Diagram *sequence* untuk skenario ini seperti pada gambar 8 :



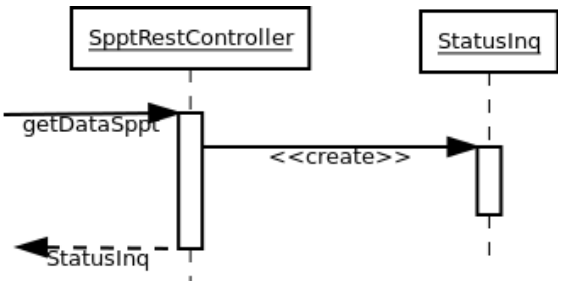
Gambar 8: Diagram *Sequence* Untuk Konfigurasi *Spring Framework*

Saat inisialisasi awal, *servlet container* akan mengeksekusi kelas AppInitializer, kemudian dilanjutkan dengan kelas AppConfig, kemudian kelas HibernateController, dan akhirnya sampai ke kelas SpptRestController.

Sampai disini, tahap inisialisasi sistem *web service* selesai, untuk selanjutnya melakukan skenario-skenario selanjutnya.

4.5.2 Skenario *Inquiry* Gagal Karena Tahun Pajak Bukan Angka

Skenario ini akan menceritakan proses *inquiry* yang dilakukan oleh *client*, namun gagal karena parameter tahun pajak yang diberikan oleh *client* terdapat karakter yang bukan angka. Diagram *sequence* untuk skenario ini seperti terlihat pada gambar 9 :



Gambar 9: Diagram *Sequence* Untuk *Inquiry* Yang Gagal Karena Tahun Pajak Mengandung Karakter Bukan Angka

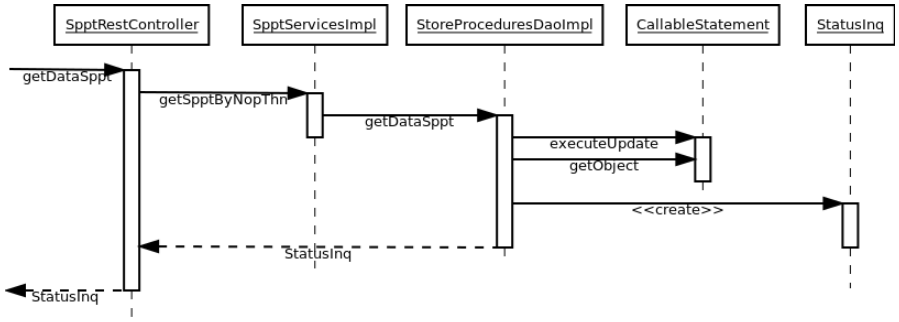
Berawal dari *method* getDataSppt milik kelas SpptRestController, karena *method* inilah yang menjadi awal tiap skenario *inquiry* terjadi, kemudian di dalam *method* ini melakukan pemeriksaan parameter tahun pajak, bila pada parameter tahun pajak terdapat karakter bukan angka, maka *method* ini akan membuat instan dari kelas StatusInq yang berisi pesan bahwa proses *request inquiry* telah gagal dilakukan karena parameter tahun pajak mengandung karakter bukan angka.

Instan dari kelas StatusInq akan diubah menjadi format JSON secara otomatis oleh pustaka Jackson, dan dikirimkan ke *client* sebagai respon. Sampai disini proses selesai.

Bila parameter tahun pajak telah valid, maksudnya tidak mengandung karakter bukan angka, maka proses akan dilanjutkan ke skenario-skenario *inquiry* yang lain.

4.5.3 Skenario *Inquiry* Gagal Karena Data Tidak Ditemukan

Skenario ini akan menjelaskan proses *inquiry* yang gagal karena data tidak ditemukan pada sistem basis data. Diagram *sequence* untuk menjelaskan skenario ini adalah seperti pada gambar 10 :



Gambar 10: Diagram *Sequence* Untuk Skenario *Inquiry* Yang Gagal Karena Data Tidak Ditemukan

Awalnya *request inquiry* dari *client* akan diterima oleh *method* getDataSppt milik kelas SpptRestController, kemudian memanggil *method* getSpptByNopThn milik kelas SpptServicesImpl sebagai tempat pemrosesan data *inquiry*, kemudian *method* getSpptByNopThn ini memanggil *method* getDataSppt milik kelas StoreProceduresDaoImpl untuk melakukan pengambilan data pada basis data.

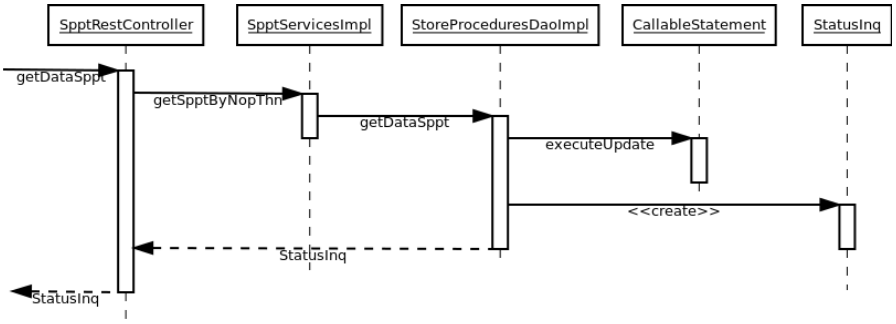
Method getDataSppt milik kelas StoreProceduresDaoImpl kemudian melakukan eksekusi terhadap *store procedure* milik sistem basis data melalui *method* executeUpdate milik kelas CallableStatement, kemudian mengambil hasilnya dari *method* getObject milik kelas yang sama.

Hasil dari pemanggilan *store procedure* milik basis data ini sesuai skenario, datanya tidak ada pada basis data, sehingga dalam *method* getDataSppt milik kelas StoreProceduresDaoImpl akan membuat instan kelas StatusInq dan mengisikan informasi bahwa data yang diinginkan tidak ada pada basis data, kemudian mengembalikan instan kelas StatusInq ke kelas SpptRestController sebagai bahan

respon ke *client*.

4.5.4 Skenario *Inquiry* Gagal Karena Kesalahan *Server*

Skenario ini akan menjelaskan proses *inquiry* yang gagal karena terjadi kesalahan pada *server*. Diagram *sequence* untuk menggambarkan skenario ini seperti terlihat pada gambar 11 :



Gambar 11: Diagram *Sequence* Untuk Skenario *Inquiry* Yang Gagal Karena Kesalahan *Server*

Request dari *client* nantinya akan masuk ke *method* getDataSppt milik kelas SpptRestController, dari sini akan memanggil kembali *method* getSpptByNopThn milik kelas SpptServicesImpl sebagai tempat pengolah data *inquiry*.

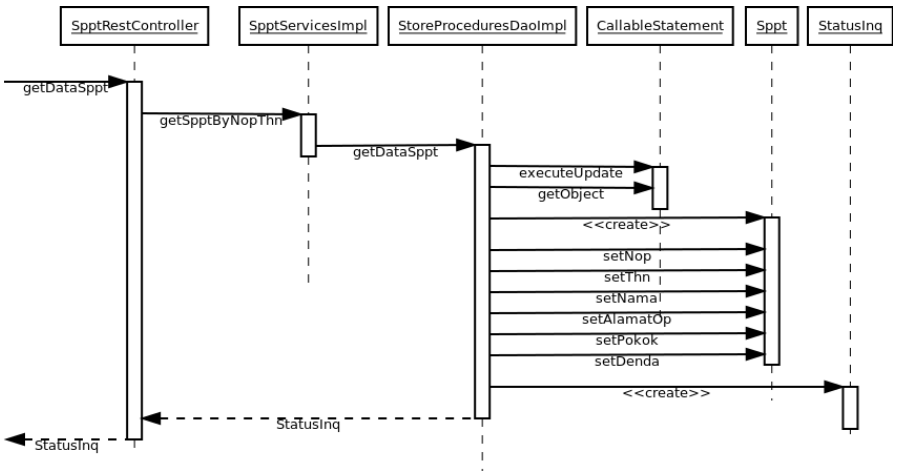
Dalam *method* getSpptByNopThn milik kelas SpptServicesImpl akan memanggil *method* yang lain yaitu getDataSppt milik kelas StoreProceduresDaoImpl yang akan melakukan tugasnya untuk berkomunikasi dengan basis data.

Dalam *method* getDataSppt milik kelas StoreProceduresDaoImpl akan melakukan eksekusi terhadap *store procedure* milik sistem basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement, namun komunikasi ini mengalami gangguan, sehingga *method* getDataSppt akan membuat instan dari kelas StatusInq dan memberikan informasi bahwa komunikasi antara *server* aplikasi dengan *server* basis data mengalami kegagalan.

Akhirnya instan kelas StatusInq ini dikembalikan ke kelas SpptRestController dan diproses sebagai respon ke *client*. Sampai disini skenario selesai.

4.5.5 Skenario *Inquiry* Yang Sukses

Skenario ini akan menjelaskan proses yang terjadi saat *request inquiry* data berhasil dilakukan dan *client* mendapatkan hasilnya. Diagram *sequence* untuk menggambarkan aktivitas ini seperti terlihat pada gambar 12 :



Gambar 12: Diagram *Sequence* Untuk Skenario *Inquiry* Yang Berhasil Dilakukan

Berawal dari proses *request* yang masuk ke *method* getDataSppt milik kelas SpptRestController, kemudian beralih ke *method* getSpptByNopThn milik kelas SpptServicesImpl yang bertugas mengolah dan memproses data *inquiry*, karena tidak ada yang diproses, maka hanya memanggil *method* getDataSppt milik kelas StoreProceduresDaoImpl.

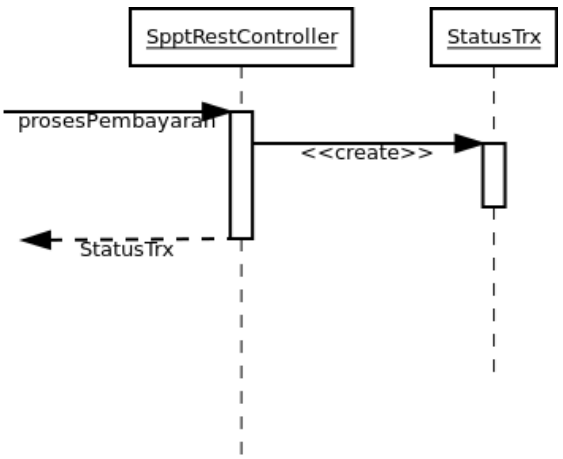
Dalam *method* getDataSppt milik kelas StoreProceduresDaoImpl akan melakukan eksekusi terhadap *store procedure* pada basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement, dan mengambil hasilnya dengan *method* getObject milik kelas yang sama.

Hasil dari pengambilan data melalui *store procedure* ini kemudian disimpan dalam instan dari kelas Sppt dengan melakukan pemanggilan *method* set pada propertinya. Terakhir adalah membuat instan kelas StatusInq dan memasukkan informasi bahwa *request inquiry* telah berhasil dan memasukan instan kelas Sppt ke dalam instan kelas StatusInq.

Sebagai respon ke *client*, maka instan kelas StatusInq dikembalikan ke kelas SpptRestController untuk selanjutnya diubah dalam format JSON oleh pustaka Jackson.

4.5.6 Skenario Transaksi Pembayaran Gagal Karena Jam Pembayaran Melebihi Jam Pencatatan

Skenario ini akan menceritakan interaksi yang terjadi antar kelas bila ada kondisi pembayaran yang gagal karena parameter jam dan tanggal pembayaran yang diberikan oleh *client* lebih baru atau melewati waktu saat ini. Diagram *sequence* yang menggambarkan skenario ini seperti terlihat pada gambar 13 :



Gambar 13: Diagram *Sequence* Untuk Skenario Transaksi Pembayaran Yang Gagal Karena Jam Pembayaran Melebihi Jam Pencatatan

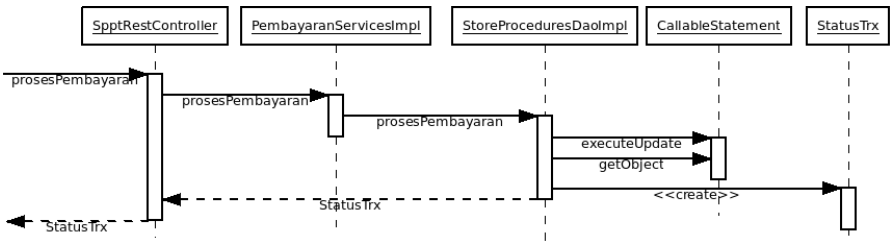
Request pencatatan pembayaran yang datang dari *client* akan masuk ke *method*

prosesPembayaran milik kelas SpptRestController yang kemudian dilakukan pemeriksaan parameter tanggal pembayaran ditemukan bahwa parameter tersebut melebihi tanggal dan jam saat ini, maka *method* prosesPembayaran dalam kelas SpptRestController ini langsung membuat instan dari kelas StatusTrx dan memberikan informasi bahwa parameter tanggal dan jam pembayaran telah melewati tanggal dan jam pencatatan saat ini.

Instan dari kelas StatusTrx secara otomatis diubah ke format JSON oleh pustaka Jackson sebagai bentuk respon ke *client*. Sampai disini skenario selesai.

4.5.7 Skenario Transaksi Pembayaran Gagal Karena Tagihan Telah Terbayar atau Nihil

Skenario ini akan menceritakan *request* pencatatan transaksi pembayarna yang gagal karena tagihan telah terbayar atau memang tidak ada tagihan. Diagram *sequence* yang menggambarkan skenario ini seperti terlihat pada gambar 14 :



Gambar 14: Diagram *Sequence* Untuk Skenario Transaksi Pembayaran Yang Gagal Karena Tagihan Telah Terbayar Atau Tagihan Nihil

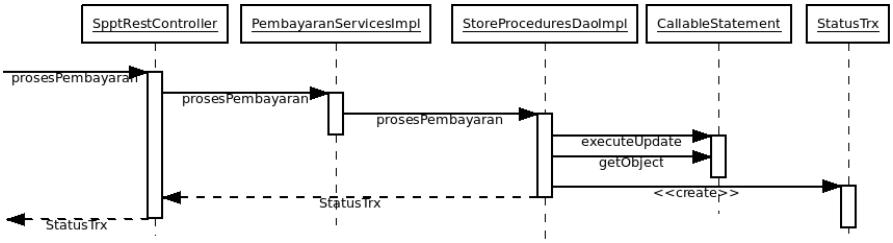
Berawal dari *method* prosesPembayaran dalam kelas SpptRestController, dilanjutkan ke pemanggilan *method* prosesPembayaran dalam kelas PembayaranServicesImpl yang fungsinya untuk mengolah data pembayaran, selanjutnya berlanjut ke *method* prosesPembayaran dalam kelas StoreProceduresDaoImpl yang akan berhubungan dengan sistem basis data.

Dalam *method* prosesPembayaran milik kelas StoreProceduresDaoImpl akan melakukan eksekusi *store procedure* milik sistem basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement, dan mengambil hasilnya dengan *method* getObject milik kelas yang sama.

Hasil dari eksekusi *store procedure* ini menghasilkan status bahwa data untuk Nomor Objek Pajak (NOP) pada tahun pajak yang diinginkan tagihannya telah terbayar atau tidak ada tagihan. Sehingga langkah selanjutnya yang dilakukan dalam *method* prosesPembayaran milik kelas StoreProceduresDaoImpl akan membentuk instan dari kelas StatusTrx dan mengisikan pesan bahwa data yang diminta telah terbayar atau nihil. Instan dari kelas StatusTrx ini nantinya dikirimkan ke kelas SpptRestController sebagai bahan *respon* ke *client*. Sampai disini skenario selesai.

4.5.8 Skenario Transaksi Pembayaran Gagal Karena Telah Dibatalkan

Skenario ini akan menjelaskan alur komunikasi bila ada *request* transaksi pembayaran tetapi gagal karena data tagihan telah dibatalkan. Diagram *sequence* untuk skenario ini seperti terlihat pada gambar 15 :



Gambar 15: Diagram *Sequence* Untuk Skenario Transaksi Pembayaran Yang Gagal Karena Tagihan Telah Dibatalkan

Diawali dari *method* prosesPembayaran pada kelas SpptRestController, karena setiap *request* yang masuk untuk melakukan pencatatan transaksi pembayaran

akan melalui *method* ini. Kemudian dilakukan pemanggilan *method* prosesPembayaran milik kelas PembayaranServicesImpl, dilanjutkan dengan pemanggilan *method* prosesPembayaran milik kelas StoreProceduresDaoImpl.

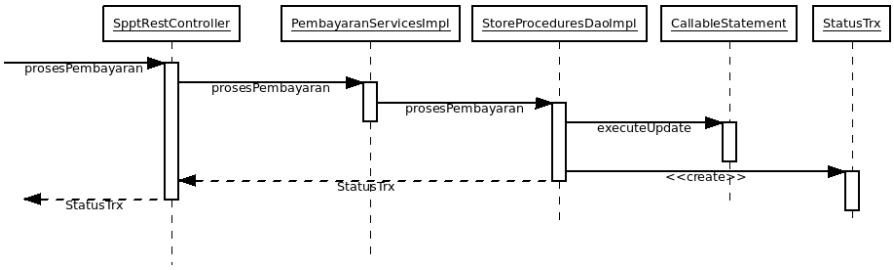
Di dalam *method* prosesPembayaran milik kelas StoreProceduresDaoImpl, akan melakukan eksekusi terhadap *store procedure* milik sistem basis data dengan menggunakan *method* executeUpdate dan mengambil hasilnya dengan *method* getObject.

Karena hasil yang diterima dari basis data adalah bahwa tagihan untuk Nomor Objek Pajak (NOP) pada tahun pajak yang diinginkan telah dibatalkan, maka di dalam *method* prosesPembayaran milik kelas StoreProceduresDaoImpl ini membuat instan dari kelas StatusTrx dan mengisikan dengan informasi tersebut.

Kemudian kelas StatusTrx ini dikirimkan ke kelas SpptRestController sebagai bahan respon ke *client*.

4.5.9 Skenario Transaksi Pembayaran Gagal Karena Kesalahan *Server*

Skenario ini akan menceritakan kondisi *request* pencatatan transaksi pembayaran karena telah terjadi kesalahan di sisi *server*. Diagram *sequence* yang menggambarkan skenario ini seperti terlihat pada gambar 16 :



Gambar 16: Diagram *Sequence* Untuk Skenario Transaksi Pembayaran Yang Gagal Karena Adanya Kesalahan *Server*

Seperti biasa, diawali dari *method* prosesPembayaran milik kelas SpptRest-

Controller, yang didalamnya memanggil *method* prosesPembayaran milik kelas PembayaranServicesImpl sebagai tempat untuk memproses atau mengolah data transaksi pembayaran.

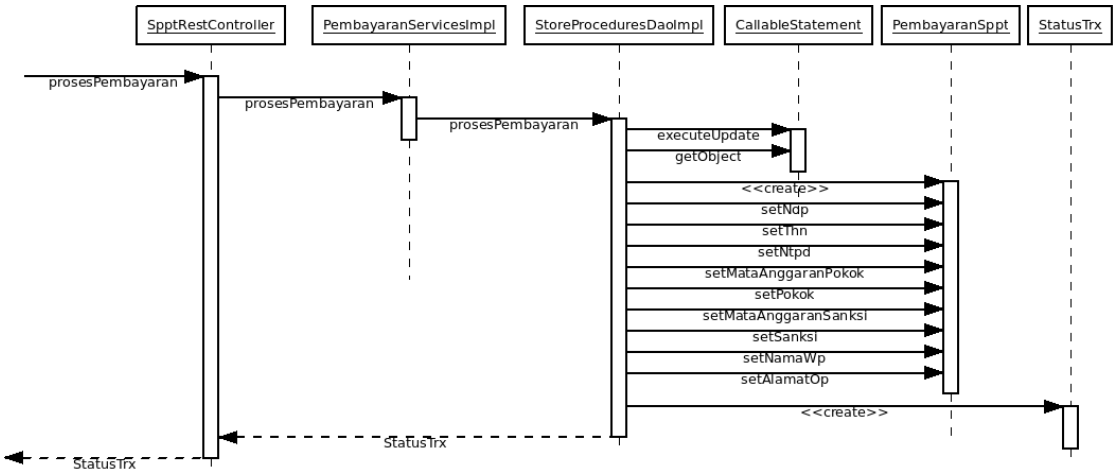
Di dalam *method* prosesPembayaran milik kelas PembayaranServicesImpl akan memanggil *method* prosesPembayaran milik kelas StoreProceduresDaoImpl yang bertugas melakukan komunikasi dengan sistem basis data.

Dalam *method* prosesPembayaran milik kelas StoreProceduresDaoImpl akan melakukan eksekusi *store procedure* milik sistem basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement. Namun pada prosesnya terjadi kegagalan komunikasi dengan sistem basis data, sehingga langkah selanjutnya dalam *method* prosesPembayaran milik kelas StoreProceduresDaoImpl akan membentuk instan kelas StatusTrx dan memberikan informasi di dalamnya bahwa komunikasi dengan *server* basis data telah terjadi kegagalan sehingga *request* dibatalkan.

Instan kelas StatusTrx kemudian dikirimkan ke kelas SpptRestController sebagai bahan respon ke *client*. Sampai disini skenario transaksi pembayaran yang gagal karena kesalahan *server* selesai.

4.5.10 Skenario Transaksi Pembayaran Yang Sukses

Skenario ini akan menjelaskan bagaimana alur komunikasi untuk *request* transaksi pembayaran yang berhasil. Diagram *sequence* untuk skenario ini seperti terlihat pada gambar 17 :



Gambar 17: Diagram *Sequence* Untuk Skenario Transaksi Pembayaran Yang Sukses

Seperti biasanya, berawal dari *method* prosesPembayaran milik kelas SpptRestController, kemudian memanggil *method* prosesPembayaran milik kelas PembayaranServicesImpl, kemudian di dalamnya memanggil *method* prosesPembayaran milik kelas StoreProceduresDaoImpl.

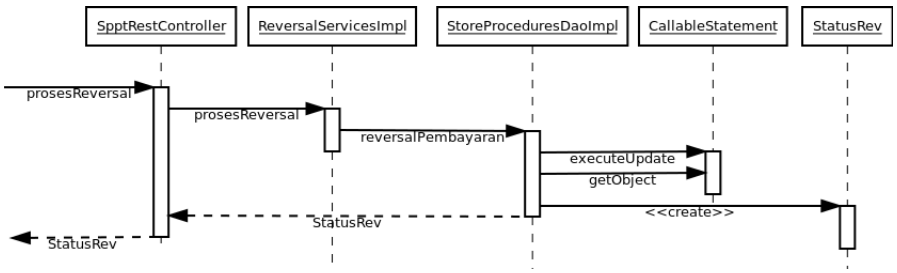
Pada *method* prosesPembayaran milik kelas StoreProceduresDaoImpl kemudian melakukan eksekusi terhadap *store procedure* milik sistem basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement dan mengambil hasilnya dengan memanggil *method* getObject dan menyimpan hasilnya dalam instan kelas PembayaranSppt.

Kemudian instan dari kelas PembayaranSppt dimasukkan dalam instan kelas StatusTrx yang kemudian dikirimkan ke kelas SpptRestController sebagai bahan respon ke *client*.

4.5.11 Skenario *Reversal* Gagal Karena Data Yang Diminta Tidak Ada

Skenario ini akan menceritakan alur komunikasi yang terjadi saat ada *request reversal* dari *client* tetapi gagal karena data yang diminta tidak ada pada sistem

basis data. Diagram *sequence* untuk menggambarkan skenario ini seperti terlihat pada gambar 18 :



Gambar 18: Diagram *Sequence* Untuk Skenario *Reversal* Yang Gagal Karena Data Yang Diminta Tidak Ada

Berawal dari *method* prosesReversal milik kelas SpptRestController, kemudian dilanjutkan dengan pemanggilan *method* prosesReversal milik kelas ReversalServicesImpl yang bertugas melakukan proses data berkaitan dengan *reversal* pembayaran.

Dalam *method* prosesReversal milik kelas ReversalServicesImpl akan memanggil *method* reversalPembayaran milik kelas StoreProceduresDaoImpl yang bertugas melakukan komunikasi dengan sistem basis data.

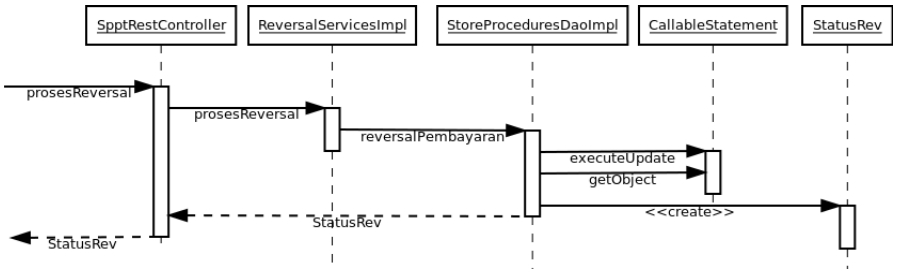
Pada *method* reversalPembayaran milik kelas StoreProceduresDaoImpl, aplikasi akan melakukan eksekusi terhadap *store procedure* pada sistem basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement, dan mendapatkan hasilnya dengan memanggil *method* getObject milik kelas yang sama.

Namun atas dasar parameter yang diberikan oleh *client*, ternyata pada basis data tidak ada data dengan kriteria tersebut sehingga dalam *method* reversalPembayaran milik kelas StoreProceduresDaoImpl langsung membuat instan dari kelas StatusRev dengan memberikan keterangan bahwa data yang diminta untuk dilakukan proses *reversal* tidak ada pada basis data, dan mengirimkan instan dari kelas StatusRev ini ke kelas SpptRestController sebagai bahan respon ke *client*.

Sampai disini proses dari skenario ini selesai.

4.5.12 Skenario *Reversal* Gagal Karena Ada Data Pembayaran Yang Tercatat Ganda

Skenario ini akan menceritakan alur komunikasi dari proses *reversal* yang gagal karena data pembayaran tercatat ganda pada basis data. Diagram *sequence* untuk skenario ini sama persis dengan skenario sebelumnya, yaitu skenario *reversal* yang gagal karena data yang diminta tidak ada pada basis data. Diagram *sequence* untuk menggambarkan proses dari skenario ini adalah seperti pada gambar 19 :



Gambar 19: Diagram *Sequence* Untuk Skenario *Reversal* Yang Gagal Karena Ada Data Pembayaran Yang Tercatat Ganda

Berawal dari *method* prosesReversal milik kelas SpptRestController yang didalamnya memanggil *method* prosesReversal milik kelas ReversalServicesImpl, dalam *method* ini akan memanggil *method* reversalPembayaran milik kelas StoreProceduresDaoImpl sebagai *method* yang bertugas melakukan komunikasi dengan basis data.

Pada *method* reversalPembayaran milik kelas StoreProceduresDaoImpl akan melakukan eksekusi terhadap *store procedure* milik basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement dan mengambil hasilnya dengan memanggil *method* getObject milik kelas yang sama.

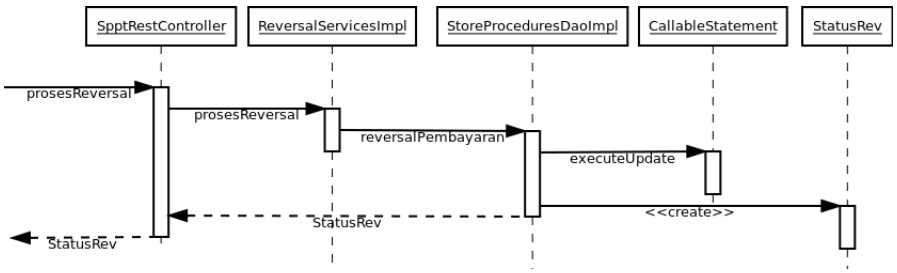
Perbedaan dengan skenario sebelumnya adalah pada saat pengambilan hasil

dari eksekusi *store procedure*, pada skenario ini *store procedure* akan memberikan informasi bahwa transaksi pembayaran untuk data yang diminta tercatat 2 (dua) kali pada basis data. Sehingga pada *method* reversalPembayaran milik kelas StoreProceduresDaoImpl akan membuat instan dari kelas StatusRev dan memberikan catatan informasi didalamnya bahwa proses *reversal* yang diminta tidak dapat diselesaikan karena ada pencatatan pembayaran ganda pada basis data.

Kemudian instan dari kelas StatusRev ini dikirimkan ke kelas SpptRestController sebagai bahan respon ke *client*, sampai disini proses untuk skenario ini selesai.

4.5.13 Skenario *Reversal* Gagal Karena Kesalahan *Server*

Skenario ini akan menceritakan proses *reversal* yang gagal karena terjadi kesalahan di sisi *server*. Diagram *sequence* yang menggambarkan skenario ini seperti terlihat pada gambar 20 :



Gambar 20: Diagram *Sequence* Untuk Skenario *Reversal* Yang Gagal Karena Ada Kesalahan *Server*

Seperti biasa, proses *reversal* akan diawali dari *method* prosesReversal milik kelas SpptRestController, yang di dalamnya akan memanggil *method* prosesReversal milik kelas ReversalServicesImpl, yang di dalamnya akan memanggil *method* lain yaitu *method* reversalPembayaran milik kelas StoreProceduresDaoImpl sebagai *method* yang akan menghubungkan atau mengeksekusi *store procedure* pada

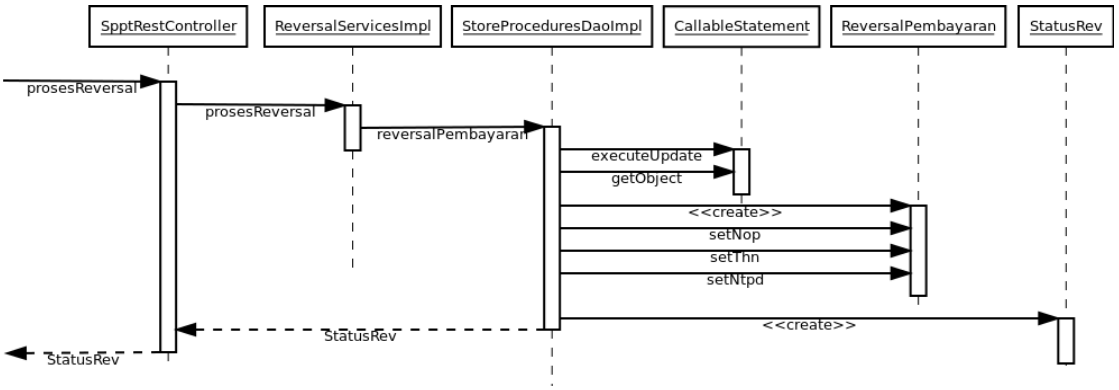
basis data.

Untuk melakukan eksekusi *store procedure* pada sistem basis data, di dalam *method* reversalPembayaran milik kelas StoreProceduresDaoImpl akan memanggil *method* executeUpdate milik kelas CallableStatement, namun karena terjadi kesalahan komunikasi antara *server* aplikasi dengan *server* basis data, maka prosesnya akan terhenti sehingga *method* reversalPembayaran milik kelas StoreProceduresDaoImpl akan membuat instan dari kelas StatusRev dan mengisikannya dengan informasi kegagalan komunikasi dengan *server* basis data.

Kemudian instan dari kelas StatusRev ini dikirimkan ke kelas SpptRestController sebagai bahan respon ke *client*. Sampai disini proses dari skenario ini telah selesai.

4.5.14 Skenario *Reversal* Yang Sukses

Skenario ini akan menceritakan alur komunikasi proses *reversal* yang sukses dilakukan. Diagram *sequence* untuk menggambarkan skenario ini adalah seperti terlihat pada gambar 21 :



Gambar 21: Diagram *Sequence* Untuk Skenario *Reversal* Yang Sukses

Diawali dari *method* prosesReversal milik kelas SpptRestController, kemudian di dalamnya memanggil *method* prosesReversal milik kelas ReversalServicesImpl.

Kemudian di dalam *method* prosesReversal milik kelas ReversalServicesImpl akan memanggil *method* reversalPembayaran milik kelas StoreProceduresDaoImpl yang bertugas melakukan komunikasi dengan basis data.

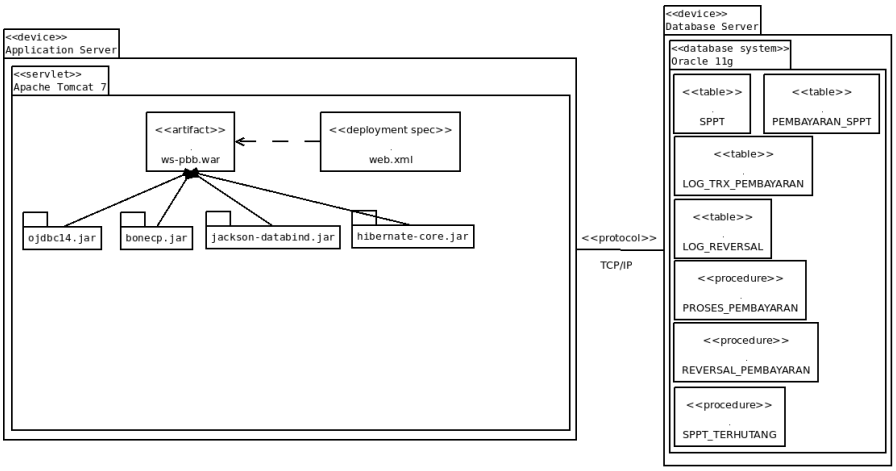
Di dalam *method* reversalPembayaran milik kelas StoreProceduresDaoImpl akan melakukan eksekusi terhadap *store procedure* pada basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement, dan mengambil hasilnya dengan memanggil *method* getObject.

Hasil yang didapat dari pemanggilan *store procedure* milik basis data akan disimpan dalam instan kelas ReversalPembayaran, yang kemudian dibuatkan pula instan kelas StatusRev dan memasukkan informasi di dalamnya bahwa proses *reversal* telah berhasil dan data yang dilakukan *reversal* ada pada kelas ReversalPembayaran yang juga dimasukkan pada instan kelas StatusRev.

Kemudian instan kelas StatusRev dikirim ke kelas SpptRestController sebagai bahan respon ke *client*, sampai disini proses dari skenario *reversal* yang sukses selesai.

4.6 Diagram *Deployment*

Diagram ini akan menjelaskan bagaimana sistem aplikasi ditempatkan, dan konfigurasi sistem seperti apa saja yang diperlukan pada perangkat keras agar sistem dapat berjalan sebagaimana mestinya. Diagram *deployment* untuk sistem aplikasi ini seperti terlihat pada gambar 22 :



Gambar 22: Diagram *Deployment*

Sebagaimana terlihat pada diagram, bahwa nantinya sistem ini yang bernama ws-pbb akan dipasangkan pada *server* aplikasi yang memiliki aplikasi server bernama Apache Tomcat versi 7.

Kebutuhan akan pustaka sebetulnya sudah secara otomatis terkait pada sistem aplikasi yang dibangun dengan menggunakan manajemen pustaka Maven. Sedangkan *file* konfigurasi web.xml terbentuk secara otomatis melalui manajemen pustaka Maven.

Dalam diagram diperlihatkan bahwa nantinya sistem aplikasi akan berinteraksi dengan sistem basis data pada *server* basis data, dimana sistem basis data yang digunakan adalah Oracle 11g, dengan tabel-tabel dan *store procedure* yang disebutkan dalam diagram.

5 SUMBER DAYA MANUSIA

Karena bentuk sistem ini adalah bentuk otomasi dari pencatatan pembayaran, dimana komunikasi yang terjadi antara *server* pada DPPK Kabupaten Brebes dengan *server* pada Bank sebagai tempat pembayaran, tidak ada sumber daya

manusia yang berinteraksi secara langsung dengan sistem ini.