DOKUMENTASI ANALISIS SISTEM *WEB SERVICES* SEBAGAI PENCATAT TRANSAKSI PEMBAYARAN PBB-P2

PERIODE PENILAIAN TAHUN 2016

Oleh :

Priyanto Tamami, S.Kom.

NIP 19840409 201001 1 025

Fungsional Pranata Komputer

Dinas Pendapatan dan Pengelolaan Keuangan

Pemerintah Daerah Kabupaten Brebes

Brebes, 8 September 2016

Lembar Pengesahan

Nama Kegiatan	:	Melaksanakan Analisis Sistem Informasi
Judul	:	DOKUMENTASI ANALISIS SISTEM *WEB SERVICES* SEBAGAI PENCATAT TRANSAKSI PEMBAYARAN PBB-P2

Disetujui oleh :	Disusun Oleh
Kepala Seksi Pendataan, Penetapan, dan Keberatan	Pranata Komputer
Pada tanggal 9 September 2016	Selesai tanggal : 8 September 2016
Fetiana Dwiningrum, SIP, M.Si.	Priyanto Tamami, S.Kom
NIP 19880223 200701 2 001	NIP 19840409 201001 1 025

DAFTAR ISI

DAFTAR GAMBAR

BAB 1

SASARAN DAN BATASAN SISTEM

Sebagai sebuah sistem yang dibangun untuk alasan atau tujuan tertentu, sistem aplikasi *web services* ini pun dibangun dengan beberapa sasaran tertentu diantaranya yaitu :

1. Menjaga konsistensi basis data SISMIOP agar data yang tersimpan dan diproduksi dari basis data ini valid tanpa perlu dilakukan pengolahan terlebih dahulu.

2. Data realisasi pembayaran PBB-P2 dapat disajikan secara *realtime* tanpa jeda hari, jam, bahkan menit.

3. Perubahan-perubahan data akibat pengajuan pelayanan dapat langsung dibayarkan detik itu juga setelah wajib pajak atau kuasanya melakukan pengambilan berkas selesai di DPPK, data tagihan terbaru langsung dapat diakses oleh Bank sebagai tempat pembayaran.

Dari sasaran yang akan dicapai tersebut, karena kondisi cakupan *web services* yang terdiri dari banyak protokol dan spesifikasi, boleh dikatakan luas cakupannya,

maka sistem yang dibangun untuk pencatatan pembayaran PBB-P2 ini sebetulnya lebih ke *Web API*, dimana *Web API* ini adalah jenis *web services* yang penekanannya telah berubah menjadi komunikasi dengan basis bentuk yang lebih sederhana dari *representational state transfer* (REST). *RESTful API* tidak memerlukan protokol *web services* berbasis XML (seperti SOAP dan WSDL) untuk mendukung *interface*-nya.

Karena berbentuk *web services* untuk melayani pencatatan pembayaran PBB-P2 oleh Bank sebagai tempat pembayaran, maka yang dibangun hanya berupa *server* yang melayani *web services*, adapun pembuatan aplikasi *client* nantinya hanya diperuntukan sebagai media untuk melakukan pengetesan saja.

BAB 2

ARSITEKTUR SISTEM

Jika membahas arsitektur sistem, maka akan ada beberapa diagram yang terbentuk untuk memudahkan dalam penjelasan sebuah sistem. Penggambaran diagram arsitektur sistem ini berkembang mengikuti model pemrogramannya, dimana dahulu ada yang dikenal dengan *flow-chart* yang menggambarkan alur proses sebuah sistem aplikasi berjalan, mulai dari awal, sampai sistem aplikasi tersebut ditutup dan selesai digunakan, karena memang model pemrograman pada saat ini berbentuk prosedural.

Namun berkembangnya jaman, dikenal istilah pemrograman berorientasi objek, yang salah satu bahasa pemrogramannya adalah Java. Dengan menggunakan Java, maka diagram *flow-chart* tidak akan bisa melakukan penggambaran alurnya karena pola pada pemrograman berorientasi objek selalu melompat dari satu kelas ke kelas yang lain, dari satu *method* ke *method* yang lain. Maka dibutuhkan penggambaran desain arsitektur yang lain selain *flow-chart*, salah satunya adalah *Unified Modelling Language* (UML).

UML sendiri sebetulnya hanya menggambarkan 2 (dua) sudut pandang dalam pemodelan sistem, yaitu :

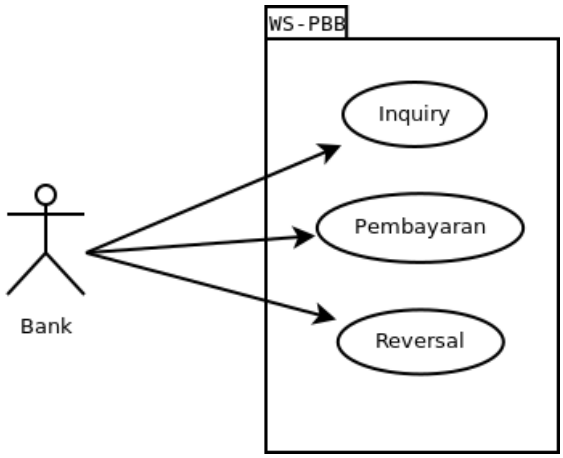
- *Static view*, yang menekankan pada struktur sistem yang bersifat statis

seperti objek, operasi, dan relasi.

- *Dynamic view*, yang menekankan pada sifat atau tingkah laku dari sistem yang menunjukkan interaksi antar objek didalamnya.

2.1 Diagram *Use-Case*

Hal yang pertama digambarkan adalah skenario penggunaan aplikasi secara umum, akan berangkat dari diagram *use-case* seperti pada gambar 2.1 :



Gambar 2.1: Diagram *use-case*

Dari diagram *use-case* diatas, skenarionya adalah bahwa Bank sebagai tempat pembayaran dapat melakukan *request* pada ketiga hal yang disediakan oleh *web services* PBB di DPPK. yaitu :

- *Inquiry*
- Pembayaran
- Reversal

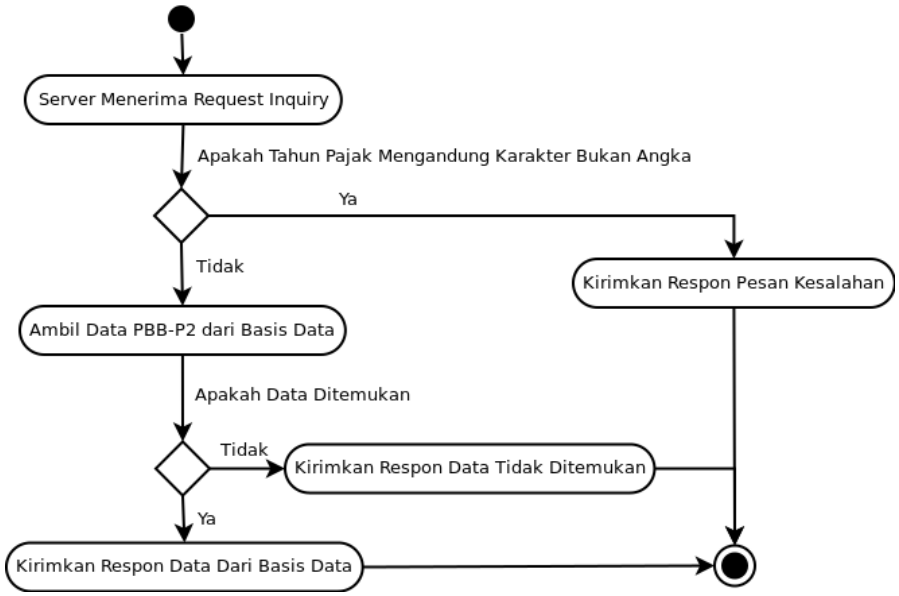
Untuk melihat masing-masing proses pada skenario diatas, ada pada diagram *activity* yang dibahas pada bagian selanjutnya.

2.2 Diagram *Activity*

Diagram *activity* ini akan menunjukan aktivitas yang terjadi untuk setiap skenario pada diagram *use-case*. Berikut skema diagram dari masing-masing skenario yang terbagi menjadi 3 (tiga) diagram berdasarkan jumlah skenario pada diagram *use-case* :

2.2.1 Diagram *Activity* Untuk *Inquiry*

Bagan diagram *activity* untuk *inquiry* ini seperti ditunjukan pada gambar 2.2 :



Gambar 2.2: Diagram *Activity* untuk *Inquiry*

Aktivitas akan dimulai dari lingkaran penuh di atas, yang kemudian didahului oleh *client* yang melakukan *request* untuk *inquiry* data PBB-P2, hal yang pertama

dilakukan adalah melakukan pemeriksaan informasi tahun pajak yang diminta, apakah mengandung karakter atau tidak.

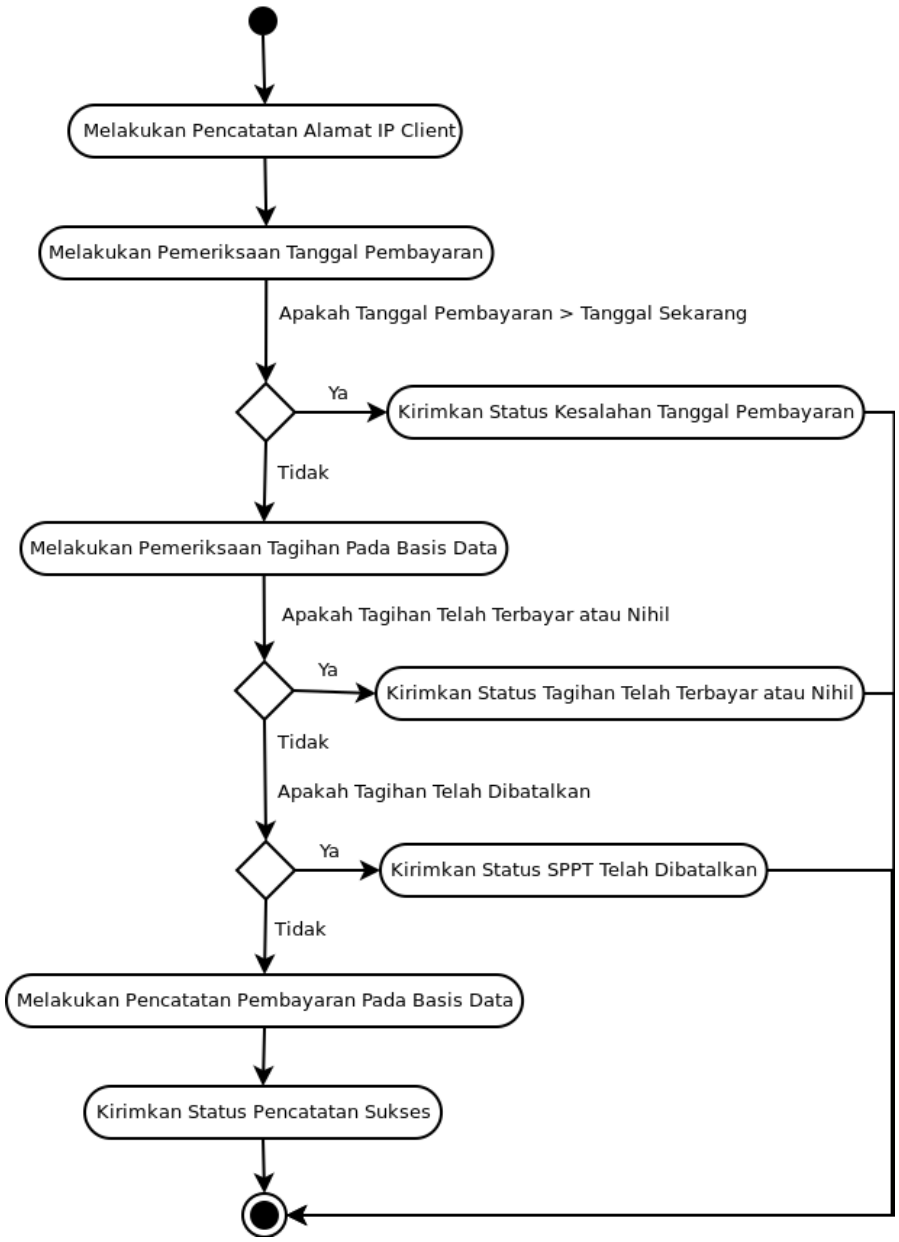
Bila Tahun pajak terisi dengan angka yang wajar, maka aplikasi *web services* akan melakukan koneksi dengan basis data SISMIOP untuk mengambil informasi-informasi yang dibutuhkan oleh *client*.

Bila data tidak ditemukan dalam basis data, maka aplikasi *web services* akan mengirimkan informasi kepada *client* bahwa data yang diminta tidak ditemukan, namun bila data ditemukan, maka disusun dalam format JSON dan dikirimkan ke *client* sebagai respon atas *request* tersebut.

Sampai sini aktivitas selesai.

2.2.2 Diagram *Activity* Untuk Pencatatan Pembayaran

Diagram *activity* untuk melakukan pencatatan pembayaran dapat dilihat seperti pada gambar 2.3 :



Gambar 2.3: Diagram *Activity* Untuk Pencatatan Pembayaran

Aktivitas pencatatan pembayaran diawali pada saat *client* melakukan *request* pembayaran SPPT PBB-P2, *server web services* akan melakukan pencatatan alamat IP darimana *request* tersebut berasal.

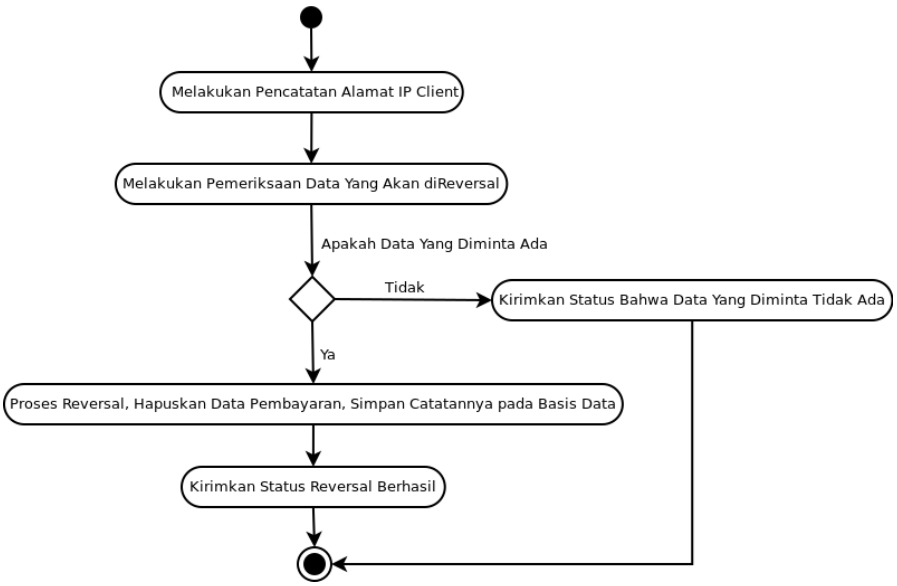
Kemudian *server* akan melakukan pemeriksaan terhadap tanggal pembayaran, apabila tanggal pembayaran melebihi tanggal saat dilakukannya proses pencatatan pembayaran, maka *server* akan mengirimkan status kesalahan tanggal pembayaran ke *client*, dan aktivitas selesai, namun bila tanggal pembayaran kurang dari atau sebelum hari dilakukannya proses pencatatan, maka aktivitas berlanjut ke proses berikutnya.

Pada tahap ini *server* melakukan pemeriksaan tagihan pada basis data, apakah kondisi objek pajak (yang teridentifikasi dari nomor objek pajak) sudah terbayar atau belum, atau tagihannya nihil (tidak ada pajak yang terhutang), bila ya, maka *server* akan mengirimkan status ke *client* bahwa objek yang diminta telah terbayar atau merupakan objek yang piutangnya nihil. bila tidak, maka proses berlanjut.

Proses akhir dari aktivitas ini adalah melakukan pencatatan pembayaran pada basis data, kemudian mengirimkan status bahwa pencatatan tersebut telah selesai. Sampai sini aktivitas telah sampai pada ujung prosesnya.

2.2.3 Diagram *Activity* Untuk *Reversal*

Diagram *activity* untuk melakukan proses *reversal* adalah sebagaimana ditunjukkan pada gambar 2.4 :



Gambar 2.4: Diagram *Activity* Untuk Proses *Reversal*

Seperti kedua aktivitas sebelumnya, hal yang pertama dilakukan adalah melakukan pencatatan alamat IP dari *client*.

Kemudian melakukan pemeriksan data yang akan direversal, terutama terhadap Nomor Transaksi Pajak Daerah (NTPD) sebagai identitas sebuah transaksi. Apabila data NTPD ini salah, maka *server* akan mengirimkan status kegagalan *reversal* karena data yang diminta tidak ada. Bila data ada, maka melanjutkan ke proses berikutnya.

Langkah berikutnya adalah melakukan pencatatan *request reversal* dan mengirimkan status ke *client* bahwa *reversal* yang diminta telah berhasil dilakukan.

Sampai sini aktivitas *reversal* selesai.

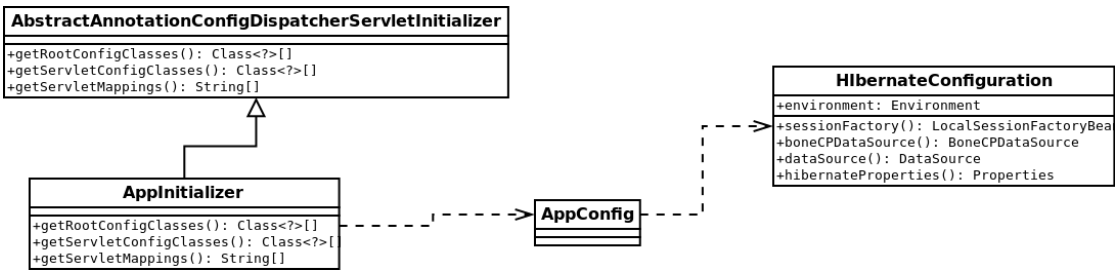
Dari diagram *use-case* dan diagram *activity* telah didapat gambaran umum dari sistem aplikasi yang akan dibangun. Diagram-diagram berikutnya akan lebih detail membahas teknis bagaimana sistem bekerja.

2.3 Diagram *Class*

Pada diagram *class* ini, akan membahas detail dari tiap kelas pembentuk sistem aplikasi *web services* secara keseluruhan. Karena akan menggunakan *framework* Spring yang tentunya memiliki spesifikasi tersendiri, maka untuk memperjelas gambar diagram *class* akan dibagi menjadi beberapa bagian, yaitu :

1. Bagian Konfigurasi

   Bagian ini adalah kelas-kelas pembentuk konfigurasi *framework* Spring sehingga aplikasi dapat dengan mudah dibangun tanpa perlu memikirkan detail teknis bagaimana Rest harus bekerja. Diagram *class* dari bagian konfigurasi ini seperti ditunjukkan pada gambar 2.5 :

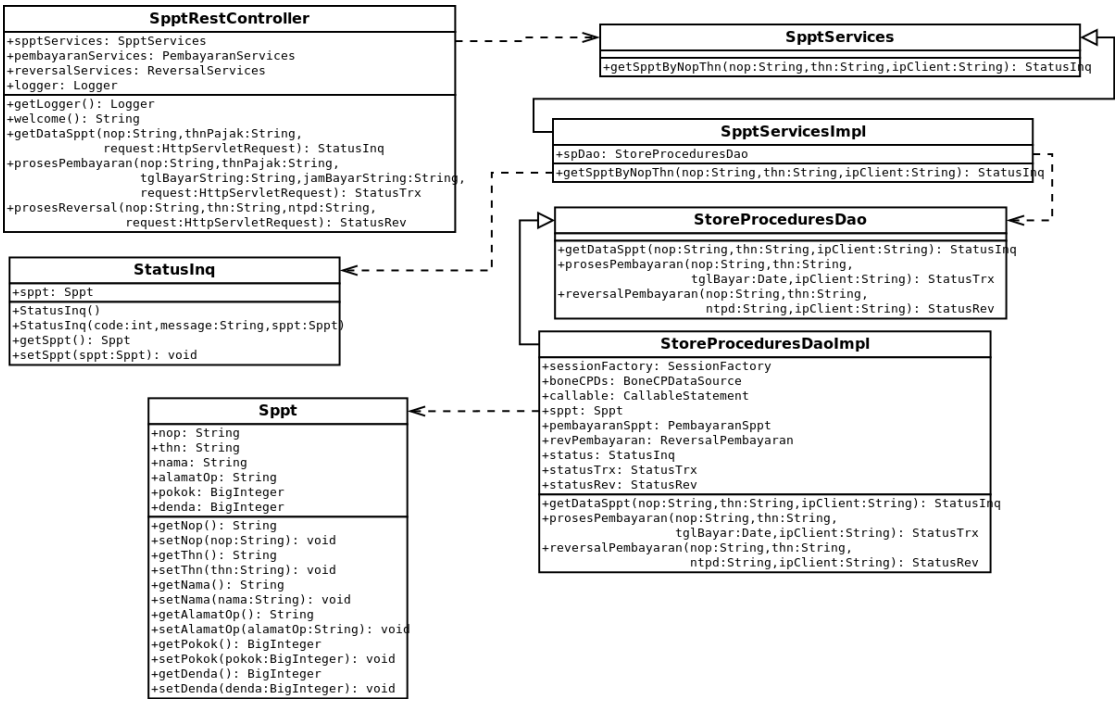


Gambar 2.5: Diagram *Class* Bagian Konfigurasi *Framework* Spring

Kelas-kelas inilah yang nantinya mengatur *services* dan konfigurasi koneksi dengan basis data. Awalnya *servlet container* akan memanggil kelas AppInitializer hasil turunan dari kelas atau *interface* AbstractAnnotationConfigDispatcherServletInitializer. Kelas AppInitializer ini membutuhkan kelas AppConfig untuk melakukan Konfigurasi khusus terkait struktur *framework* Spring yang digunakan. Kelas AppConfig ini pun melakukan pemanggilan kelas HibernateConfiguration untuk melakukan konfigurasi komunikasi dengan *server* basis data.

2. Bagian *Inquiry*

   Kelas-kelas pada bagian *inquiry* ini, beberapa akan muncul pada bagian transaksi pembayaran ataupun *reversal* karena kelas-kelas tersebut memiliki fungsi yang sama dalam siklus melayani *request* dari *client*. Kelas-kelas pada bagian *inquiry* ini akan terlihat seperti pada gambar 2.6



Gambar 2.6: Diagram *Class* Bagian *Inquiry*

Kelas yang akan menjadi titik awal datangnya sebuah *request* dari *client* adalah kelas SpptRestController, kelas ini yang nantinya akan memilih apakah *request client* adalah *inquiry*, transaksi pembayaran, atau permohonan *reversal*. Kelas SpptRestController ini pula yang nantinya mengirimkan pada *service* yang berkaitan.

Berkaitan dengan bagian *inquiry*, maka begitu ada *request inquiry* dikirimkan oleh *client* dan diterima *server* maka kemudian dipanggil *interface*

SpptServices, yang pada struktur kali ini diimplementasikan oleh kelas SpptServicesImpl.

Karena *framework* yang digunakan adalah Spring, maka diperlukan beberapa bagian berdasarkan fungsi utamanya. Fungsi dari masing-masing kelas yang berada pada *framework* Spring yaitu :

- *Domain Object*. Kelas-kelas yang memiliki fungsi ini mewakili struktur data persis seperti pada basis data. Penggunaan kelas-kelas pada *domain object* diperlukan karena :

  1) Kode program jadi lebih mudah dimengerti dan dipelihara.
  2) Karena Java merupakan bahasa yang *strongly-typed* dan harus di-*compile* terlebih dahulu, akan memudahkan pemeriksaan *bug* pada saat *compile* dibandingkan saat *runtime* atau berjalannya aplikasi.
  3) Memisahkan lapisan data dan antarmuka / *interface*. Apabila ada perubahan skema pada basis data tetapi fitur pada tampilan antarmuka tidak berubah, maka cukup dilakukan perubahan pada lapisan data / *domain object*-nya saja.
  4) Pustaka siap pakai untuk validasi. Di Java, ada pustaka yang berguna untuk melakukan validasi, yaitu JSR-303. Terhadap validasi data yang akan dimasukkan ke dalam basis data, kita tidak perlu melakukan pengecekan seperti contoh kode berikut :

```
1       if(mydata.getId() == null)
2
```

  Melainkan cukup dengan kode berikut :

```
1       @NotNull private String Id;
2
```

- *Interface Business Services*

  Kelas-kelas pada fungsi ini hanya berupa definisi daftar fitur yang disediakan oleh aplikasi. Seluruh implementasi dikelas ini belum terisi, oleh karena itu dinamakan *interface*. Beberapa alasan penggunaan *interface* atau kelas-kelas abstrak atau tanpa implementasi ini adalah sebagai berikut :

  – Pada saat membangun sebuah aplikasi *client-server*, maka cukup dengan memberikan kelas-kelas pada *domain object* dan *interface* ini pada *programmer* yang membangun sisi *client*, tanpa perlu menyertakan implementasi dari *interface* yang biasanya cukup besar, aplikasi dapat saling berkomunikasi.

  – Pada saat ada peralihan atau perubahan implementasi perangkat lunak basis data, maka aplikasi disisi *client* tidak perlu berubah.

  – Fitur *declarative transaction* yang dimiliki Spring akan lebih optimal bekerja bila dipisahkan antara *interface* dengan implementasinya.

- *Implementasi Business Services*

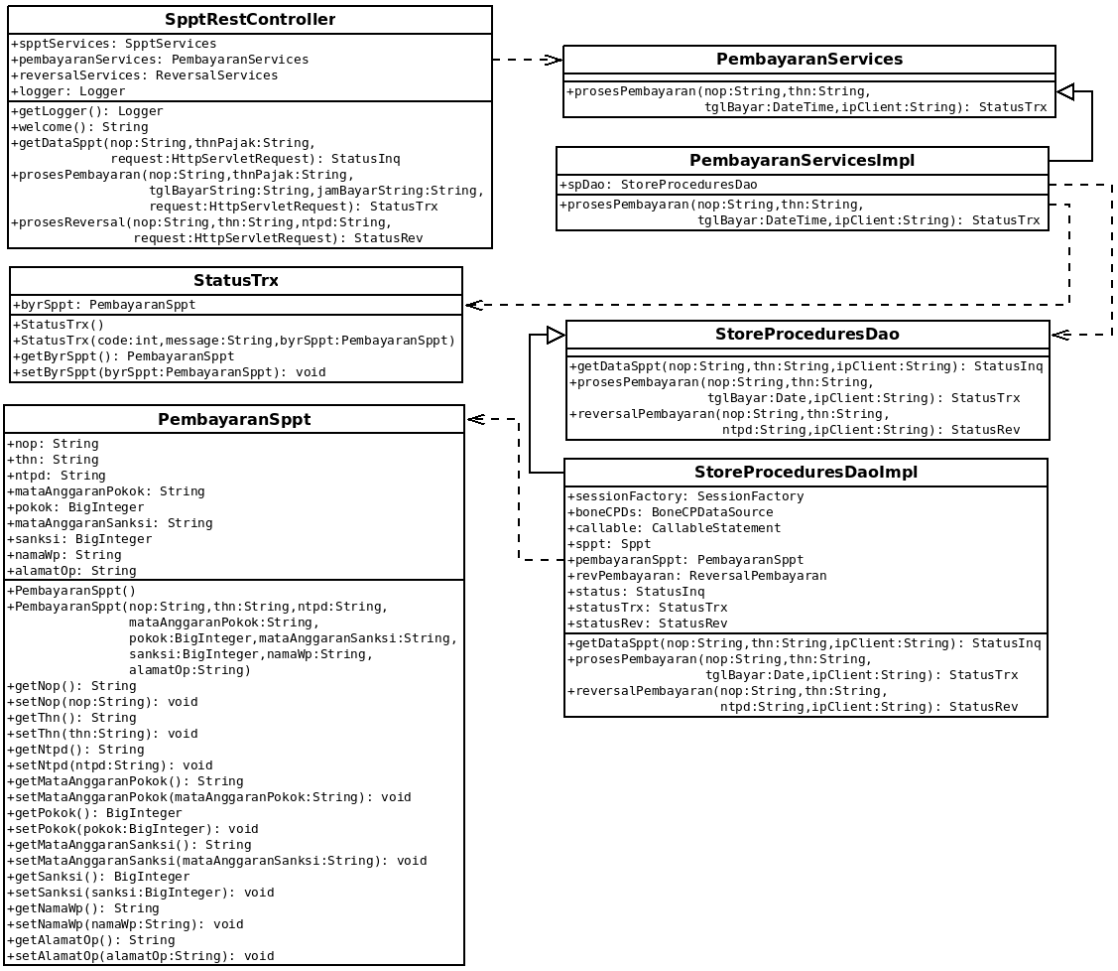
  Ini adalah bagian dari implementasi *interface business services*. Jika pada *interface* berisi fitur abstrak, pada bagian ini sudah ada implementasi konkrit untuk masing-masing fitur yang telah didefinisikan pada *interface*. Karena sistem aplikasi akan menggunakan Spring Data JPA, maka diperlukan kelas-kelas lain selain *business services*, yaitu kelas-kelas implementasi *Data Access Object* (DAO).

Implementasi dari kelas SpptServicesImpl akan mengembalikan kelas StatusInq sebagai respon terhadap *request* yang telah sampai ke *server*. Kelas SpptServicesImpl membutuhkan paket kelas DAO berupa *interface* Store-

ProceduresDao untuk melakukan komunikasi dengan basis data. Implementasi dari *interface* StoreProceduresDao ini adalah kelas StoreProceduresDaoImpl dimana implementasi didalamnya akan menggunakan kelas Sppt untuk menyimpan atau menampung nilai-nilai yang dihasilkan dari pemanggilan *store procedure* pada basis data. Yang pada akhirnya akan dikembalikan kelas Sppt ini dalam bentuk yang terbungkus dalam kelas StatusInq.

3. Bagian Transaksi Pembayaran

   Kelas-kelas pembentuk bagian transaksi pembayaran ini adalah kelas-kelas yang saling berhubungan agar *request* transaksi pembayaran dapat diproses sempurna oleh *server*. Kelas-kelas pembentuk bagian transaksi pembayaran ini adalah seperti terlihat pada gambar 2.7 :



Gambar 2.7: Diagram *Class* Untuk Bagian Pembayaran

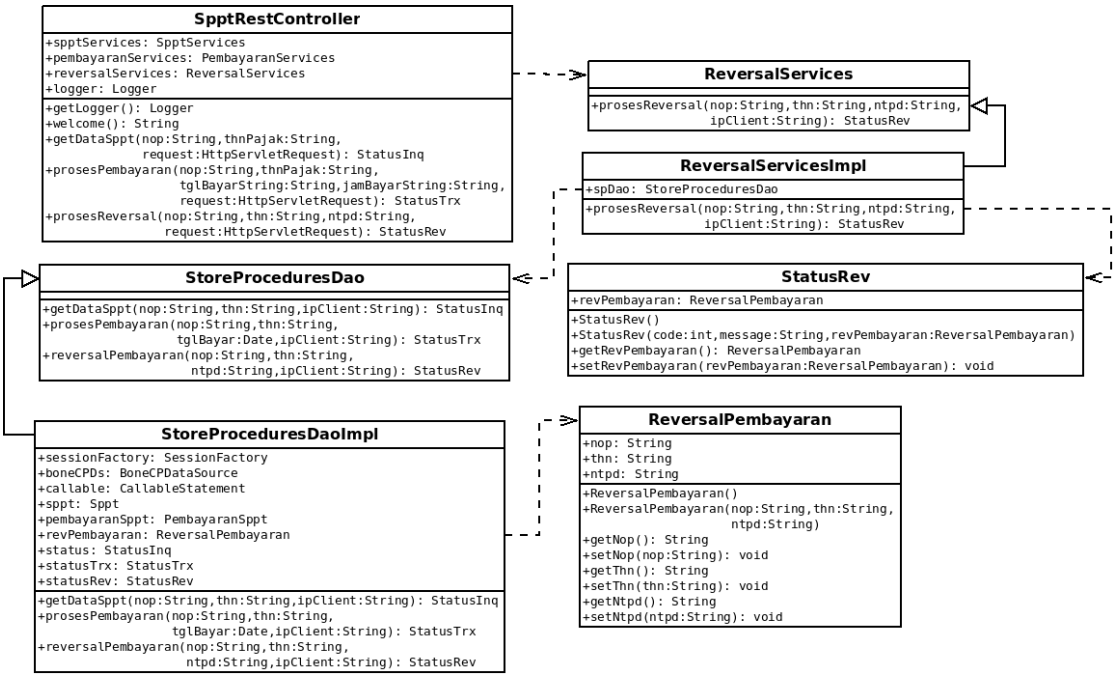
Seperti bagian sebelumnya, kelas SpptRestController yang sama muncul di bagian ini karena dari kelas inilah titik awal sebuah *request* dari *client* dimulai.

Yang berbeda dari bagian ini adalah kelas-kelas PembayaranServices dan PembayaranServicesImpl sebagai *interface* dan kelas implementasinya yang dibutuhkan oleh *framework* Spring, kemudian ada kelas StatusTrx sebagai pembungkus respon yang nantinya dikirimkan ke *client*, serta kelas PembayaranSppt untuk menampung informasi yang datang dari basis data

hasil dari pemanggilan DAO dari kelas StoreProceduresDaoImpl yang sama seperti pada bagian *inquiry*. pada bagian transaksi pembayaran ini pun, kelas PembayaranSppt sebagai bahan respon atas *request* dari *client* akan dibungkus dengan kelas *StatusTrx* agar informasi yang sampai ke *client* dapat lebih jelas.

4. Bagian *Reversal*

   Kelas-kelas pembentuk bagian *reversal* ini berfungsi untuk melakukan *reversal* transaksi yang sebelumnya telah terjadi. Kelas-kelas pada bagian ini seperti terlihat pada gambar 2.8 :



Gambar 2.8: Diagram *Class* Yang Berhubungan Dengan Proses *Reversal*

Seperti pada bagian-bagian sebelumnya, kelas SpptRestController tetap ada sebagai titik awal masuknya sebuah *request* yang akan diproses. Kemudian karena *request* yang diterima berupa proses *reversal* maka akan berhubun-

gan dengan *interface* ReversalServices dan kelas ReversalServicesImpl untuk mengikuti aturan dari *framework* Spring.

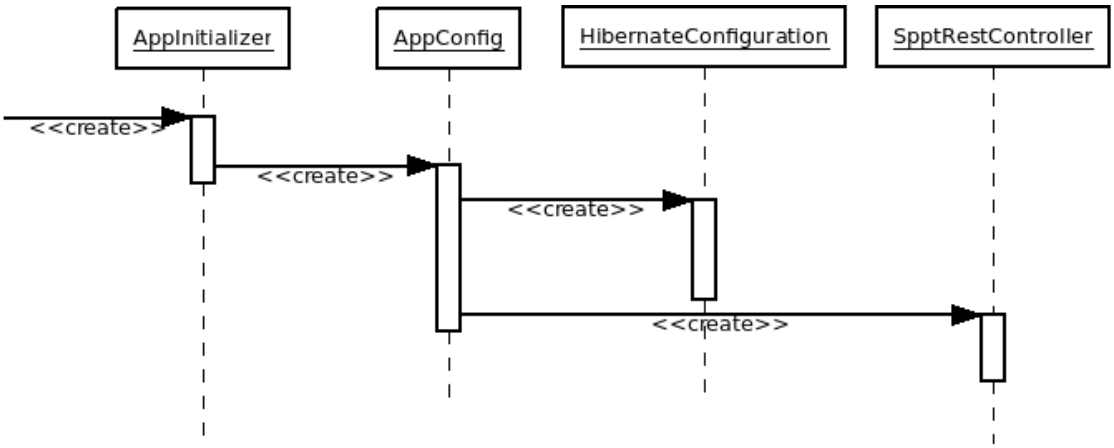
Sebagai penghubung komunikasi antara aplikasi dengan basis data, tetap menggunakan *interface* StoreProceduresDao dengan kelas StoreProcedures-DaoImpl sebagai implementasinya. Sebagai bahan respon, data-data hasil pengambilan dari basis data akan ditampung pada kelas ReversalPembayaran dengan dibungkus kelas StatusRev untuk mempermudah menambahkan informasi-informasi sukses atau gagalnya operasi yang terjadi pada *server*.

2.4 Diagram *Sequence*

Untuk penjelasan diagram *sequence* ini pun, agar diagram terlihat jelas alurnya, maka perlu dipecah menjadi beberapa bagian. Agar lebih mudah memahami prosesnya, diagram akan dipecah seperti pada diagram *class* yaitu hanya saja untuk diagram *sequence* ini akan berbentuk skenario-skenario sebagai berikut :

2.4.1 Skenario Konfigurasi *Spring Framework*

Bagian ini akan menjelaskan bagaimana alur dari awal *server* melakukan inisialisasi sistem, sampai kepada tahap *server* siap menerima *request*. Diagram *sequence* untuk keadaan ini dapat dilihat seperti pada gambar 2.9 :



Gambar 2.9: Diagram *Sequence* Untuk Konfigurasi dan Inisialisasi

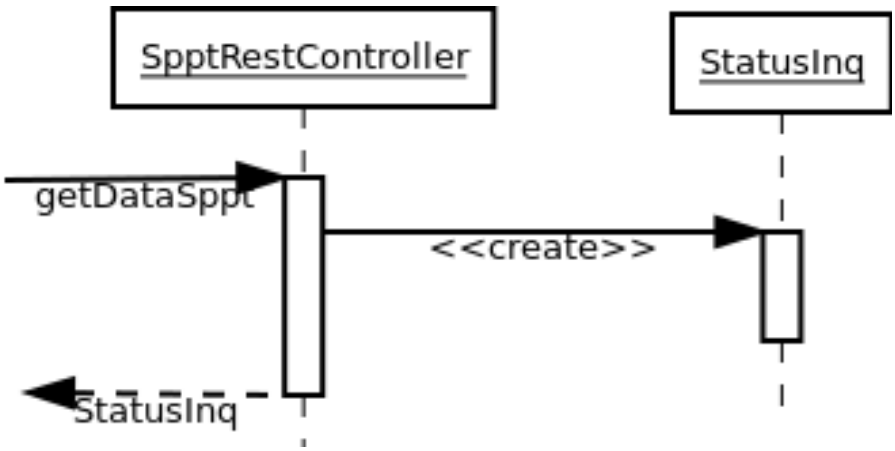
Pada awalnya *servlet container* akan melakukan inisialisasi kelas AppInitializer yang merupakan turunan dari kelas AbstractAnnotationConfigDispatcherServletInitializer, objek AbstractAnnotationConfigDispatcherServletInitializer ini merupakan *interface* yang disediakan *framework* Spring untuk tempat memulai melakukan inisialisasi.

Lalu kelas AppInitializer melakukan inisialisasi terhadap kelas AppConfig untuk melakukan konfigurasi-konfigurasi yang diperlukan agar sistem aplikasi dapat bekerja dengan baik. Salah satu diantaranya adalah melakukan inisialisasi terhadap kelas SpptRestController, kelas inilah yang nantinya melakukan seleksi *request* dan mengirimkan pada kelas *service* yang berkaitan.

Selain melakukan inisialisasi terhadap kelas SpptRestController, kelas AppConfig pun melakukan inisialisasi terhadap kelas HibernateConfiguration. Kelas HibernateConfiguration ini mempersiapkan koneksi dengan basis data dengan menyediakan modul-modul yang dapat digunakan pada sistem aplikasi nantinya.

2.4.2 Skenario *Inquiry* Gagal Karena Tahun Pajak Bukan Angka

Skenario ini akan menggambarkan kondisi dimana *client* mengirimkan *request inquiry*, namun setelah diseleksi oleh *server* terdapat karakter bukan angka pada tahun pajak yang di-*request*. Diagram *sequence* dari skenario ini dapat dilihat pada gambar 2.10 :

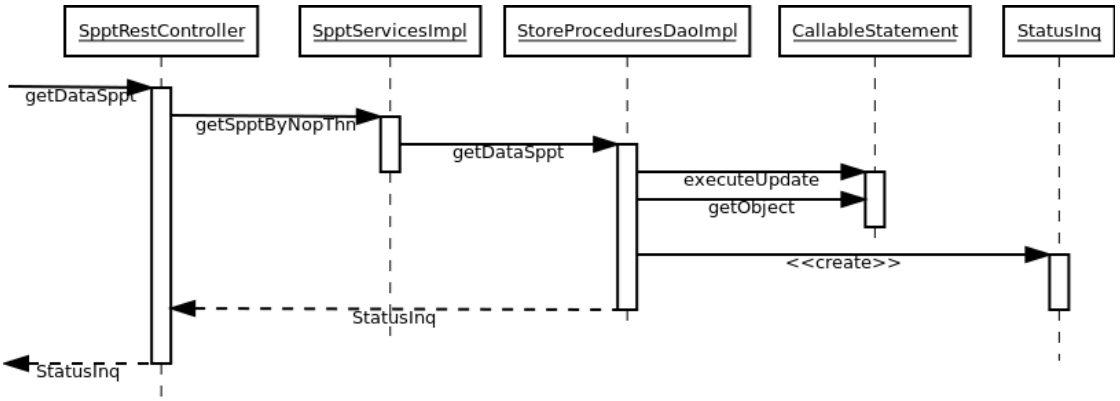


Gambar 2.10: Diagram *Sequence* Kegagalan *Inquiry* Karena Tahun Pajak Mengandung Karakter Bukan Angka

Objek yang saling berhubungan memang hanya 2 (dua) saja untuk sekenario ini, karena seleksi terhadap karakter pada parameter tahun pajak diperiksa

2.4.3 Skenario *Inquiry* Gagal Karena Data Tidak Ditemukan

Pada skenario ini, *client* mengirimkan *request* tetapi ada kesalahan bahwa tahun pajak yang di-*request* oleh *client* mengandung karakter bukan angka. Diagram *sequence* dari skenario ini seperti terlihat pada gambar 2.11 :



Gambar 2.11: Diagram *Sequence* Kegagal *Inquiry* Karena Data Tidak Ada Dalam Basis Data

Pada diagram *sequence* tersebut, permintaan atau *request* dari *client* akan diterima *server* dan diteruskan oleh *framework* Spring ke *method* getDataSppt milik kelas SpptRestController.

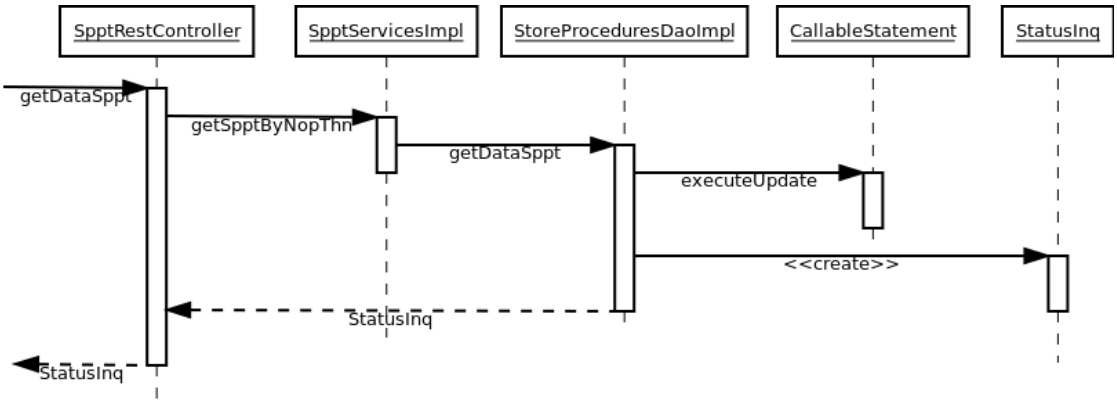
Kemudian *method* getDataSppt akan memanggil *method* getSpptByNopThn milik kelas SpptServicesImpl, yang didalamnya hanya memanggil *method* getDataSppt milik kelas StoreProceduresDaoImpl.

Didalam *method* getDataSppt milik kelas StoreProceduresDaoImpl, terdapat proses koneksi ke basis data melalui kelas CallableStatement, yang pertama melakukan pemanggilan *method* executeUpdate milik untuk mengeksekusi atau memanggil *store procedure* milik basis data, kemudian mengambil nilai balikkannya melalui *method* getObject.

Terakhir adalah melakukan pemeriksaan terhadap nilai balikan dari pembanggil *store procedure* milik basis data, bila nilai kembalian nihil, maka *method* getDataSppt akan membuat instan dari kelas StatusInq, kemudian nilainya dikembalikan ke kelas SpptRestController yang tentu saja dikirimkan hasilnya ke *client*.

2.4.4 Skenario *Inquiry* Gagal Karena Kesalahan Server

Pada skenario ini, *client* melakukan *request inquiry* ke *server*, namun karena beberapa hal, komunikasi antara *server* basis data dan *server* aplikasi mengalami gangguan, sehingga proses *inquiry* gagal dan mengirimkan pesan ke *client* bahwa *request* yang dikirimkan telah gagal diproses. Gambar Diagram *sequence* dari skenario ini dapat dilihat seperti pada gambar 2.12 :

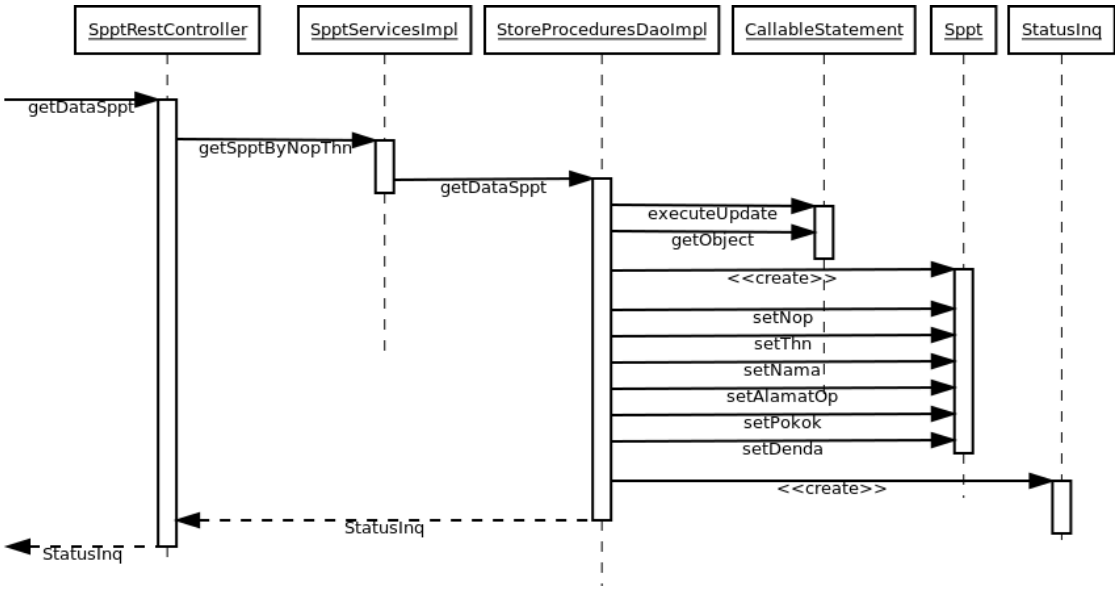


Gambar 2.12: Diagram *Sequence* Untuk Proses *Inquiry* Yang Gagal Karena Kesalahan Server

Diagramnya terlihat sama dengan skenario sebelumnya, yaitu *inquiry* yang gagal karena datanya tidak ada, yang membedakan adalah pada saat *method* getDataSppt milik kelas StoreProceduresDaoImpl melakukan pemanggilan *store procedure* milik basis data melalui *method* executeUpdate pada kelas CallableStatement ada kesalahan, sehingga *method* getDataSppt milik kelas StoreProceduresDaoImpl membuat instan dari kelas StatusInq dan mengembalikannya ke kelas SpptRestController yang kemudian menjadi respon yang dikirim ke *client* sebagai informasi bahwa *request* yang dikirimkan ke *server* gagal diproses.

2.4.5 Skenario *Inquiry* Sukses

Untuk alur proses *inquiry* yang sukses, diagram *sequence*-nya dapat dilihat seperti pada gambar 2.13 :



Gambar 2.13: Diagram *Sequence* Untuk Menangani *Request Inquiry*

Seperti awal dari setiap proses *request* Rest, pada fungsi *inquiry* ini pun berawal dari kelas SpptRestController. Kelas SpptRestController ini kemudian menyalurkannya ke kelas SpptServicesImpl dengan memanggil *method* getSpptByNopThn. Di dalam *method* getSpptByNopThn, kelas SpptServicesImpl memanggil *method* getDataSppt milik kelas StoreProceduresDaoImpl.

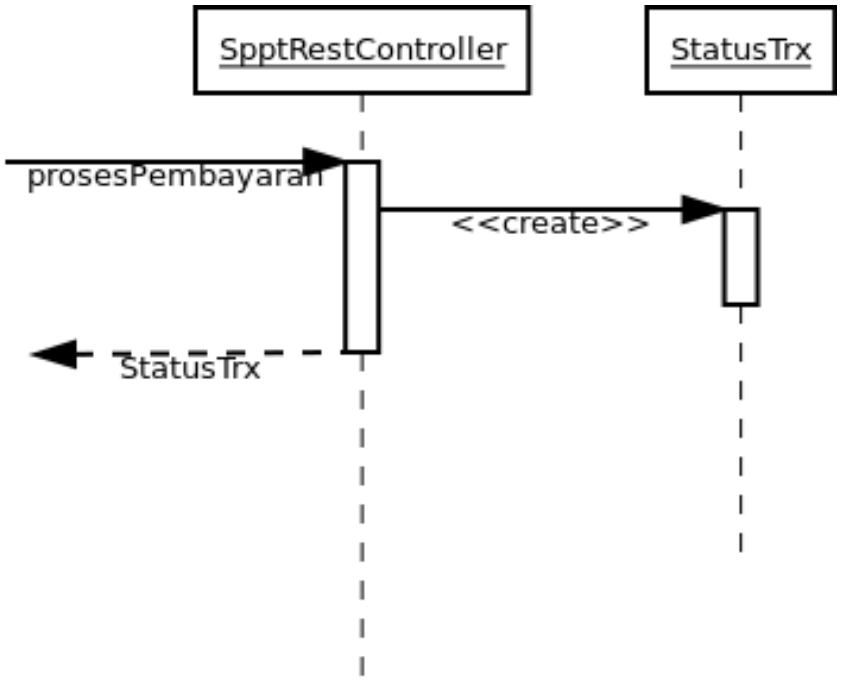
Pada kelas StoreProceduresDaoImpl, *method* getDataSppt melakukan koneksi ke basis data dan menampung hasilnya pada kelas Sppt. Hasil-hasil yang diperoleh dari basis data berupa NOP, Tahun Pajak, Nama Wajib Pajak, Alamat Objek Pajak, Besarnya tagihan pokok, dan besarnya Denda Administrasi. Semuanya ditampung yang ditandai dengan pemanggilan *method* setNop, setThn, setNama, setAlamatOp, setPokok, dan setDenda.

Sebelum dikirimkan kembali ke *client*, maka data atau informasi yang ditampung dalam kelas Sppt dipaketkan dengan kelas StatusInq, yang hasilnya dikembalikan ke *client* dalam bentuk kelas StatusInq.

2.4.6 Skenario Transaksi Pembayaran Gagal Karena Jam Pembayaran Melebihi Jam Pencatatan

Pada skenario ini, *request* transaksi yang dikirimkan oleh *client* gagal dilakukan karena waktu pada saat dilakukan pembayaran oleh wajib pajak ke tempat pembayaran, lebih dari waktu dilakukan pencatatan ke basis data. Kondisi ini tidak bisa diterima, karena seharusnya tanggal pencatatan transaksi ke basis data harus lebih terkini dibandingkan kondisi jam bahkan tanggal dilakukan pembayaran oleh wajib pajak.

Diagram *sequence* yang menggambarkan skenario ini seperti terlihat pada gambar 2.14 :



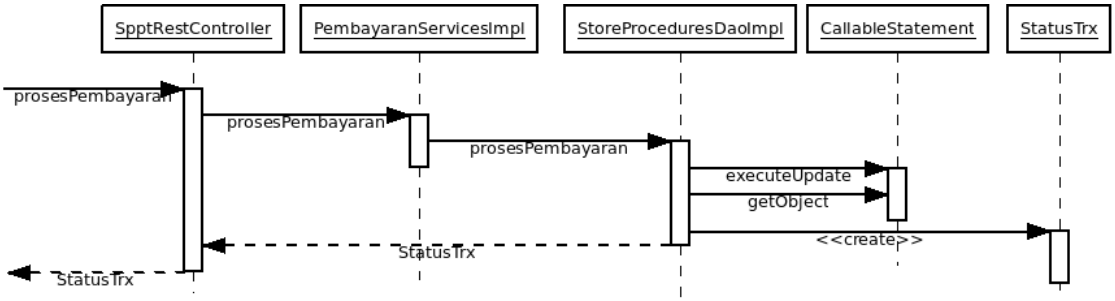
Gambar 2.14: Diagram *Sequence* Transaksi Pembayaran Yang Gagal Karena Jam Pembayaran Melebihi Jam Pencatatan

Skenario ini terlihat sederhana karena pemeriksaan tanggal pembayaran dilakukan pada *method* prosesPembayaran milik kelas SpptRestController, *method* inilah yang melakukan tugasnya pada saat ada *request* transaksi pembayaran dari *client*.

Setelah ada kesalahan tanggal pembayaran, *method* ini akan mengirimkan nilai dari kelas StatusTrx sebagai informasi bagi *client* bahwa transaksi yang dikirimkan gagal dilakukan pencatatan pembayaran karena jam dan tanggal pembayaran lebih dari tanggal dan jam sekarang.

2.4.7 Skenario Transaksi Pembayaran Gagal Karena Tagihan Telah Terbayar Atau Nihil

Skenario ini akan menggambarkan pencatatan transaksi pembayaran yang gagal dilakukan karena tagihan telah terbayar atau tagihan nihil. Diagram *sequence* dari skenario ini dapat dilihat pada gambar 2.15 :



Gambar 2.15: Diagram *Sequence* Untuk Transaksi Pencatatan Pembayaran Yang Gagal Karena Tagihan Telah Terbayar Atau Nihil

Seperti setiap *request* yang diterima dari *client* oleh *server*, bahwa *request* akan diterima oleh kelas SpptRestController, dan untuk *request* transaksi pembayaran akan ditangani oleh *method* prosesPembayaran milik kelas SpptRestController.

Dalam *method* prosesPembayaran pada kelas SpptRestController akan memanggil *method* prosesPembayaran milik PembayaranServicesImpl yang dalam *method* ini pula memanggil *method* dengan nama yang sama, yaitu prosesPembayaran, tetapi milik kelas StoreProceduresDaoImpl.

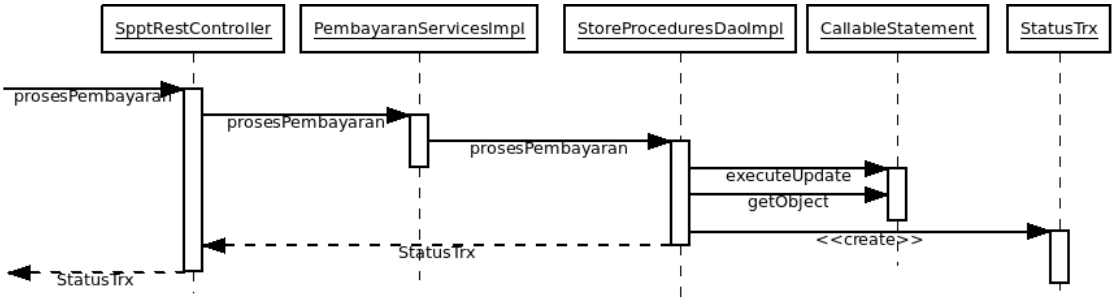
Kelas StoreProceduresDaoImpl ini akan menghubungkan aplikasi dengan *server* basis data, dengan melakukan pemanggilan pada *method* executeUpdate milik kelas *CallableStatement*, yang hasilnya dapat diambil dengan *method* getObject.

Bila hasil respon dari *server* basis data berupa kode 01, maka artinya, data tersebut telah terbayar, atau status tagihan untuk nomor objek yang diminta nihil, sehingga dalam *method* prosesPembayaran milik kelas storeProceduresDaoImpl ini

akan membuat instan dari kelas StatusTrx dan mengirimkannya kembali ke kelas SpptRestController sehingga kelas SpptRestController akan memberikan respon berupa kelas StatusTrx ke *client* sebagai informasi bahwa *request* pencatatan pembayaran telah gagal karena objek pajak yang diminta telah terbayar, atau tidak ada tagihan sama sekali.

2.4.8 Skenario Transaksi Pembayaran Gagal Karena Telah Dibatalkan

Skenario ini akan menggambarkan sebuah *request* pencatatan transaksi pembayaran dari *client* gagal karena objek pajak yang diminta status penagihannya telah dibatalkan. Diagram *sequence* untuk skenario ini dapat dilihat pada gambar 2.16 :



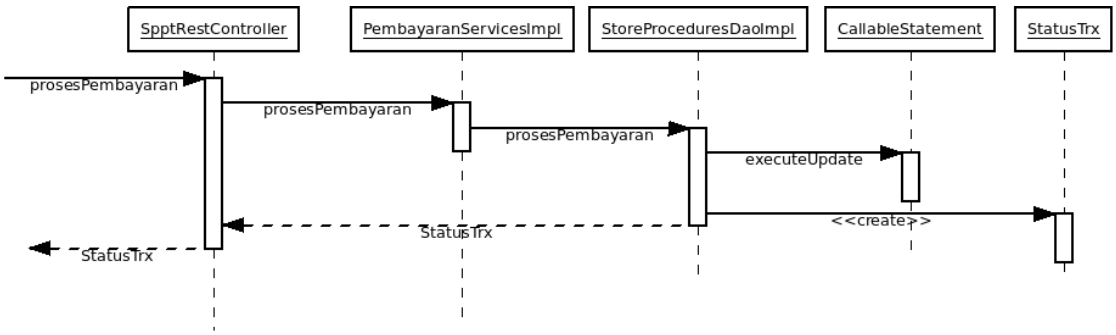
Gambar 2.16: Diagram *Sequence* Untuk Transaksi Pembayaran Yang Gagal Karena Penagihan Atas Objek Pajak Telah Dibatalkan

Terlihat mirip dengan skenario sebelumnya, dan memang *sequence* menggambarkan kondisi yang sama, yang membedakan adalah pada saat pemanggilan *method* getObject dari kelas CallableStatement, *server* basis data akan memberikan kode '04' yang artinya data penagihan untuk objek pajak yang diminta telah dibatalkan.

Tentunya *client* mengetahui hal tersebut, karena *server* akan mengirimkan kelas StatusTrx yang berisi informasi kegagalan melalui kelas SpptRestController.

2.4.9 Skenario Transaksi Pembayaran Gagal Karena Kesalahan Server

Pada skenario ini, pesan transaksi yang diminta oleh *client* sampai ke *server*, namun pada saat *server* akan melakukan komunikasi dengan basis data terjadi kegagalan. Skenario kegagalan ini dapat dilihat pada gambar 2.17 :



Gambar 2.17: Diagram *Sequence* Untuk Pencatatan Transaksi Pembayaran Yang Gagal Karena Kesalahan *Server*

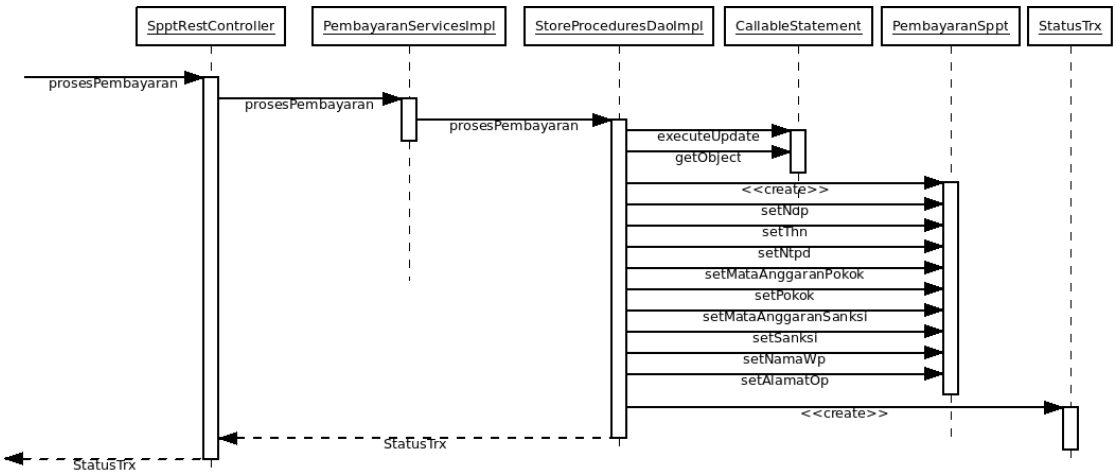
Transaksi ini diawal dari *request client* yang masuk ke *method* prosesPembayaran milik kelas SpptRestController. Kemudian secara berurutan memanggil *method* prosesPembayaran milik kelas PembayaranServicesImpl dan *method* prosesPembayaran milik kelas StoreProceduresDaoImpl.

Pada *method* prosesPembayaran milik kelas StoreProceduresDaoImpl, saat melakukan pemanggilan *store procedures* dengan *method* executeUpdate milik kelas CallableStatement, terjadi kesalahan komunikasi antara *server* aplikasi dengan *server* basis data, sehingga proses terlempar ke *exception* dan dilakukan pembuatan instan kelas StatusTrx oleh kelas StoreProceduresDaoImpl, dan dikembalikan

ke kelas SpptRestController dalam bentuk kelas StatusTrx dan disampaikan ke *client*.

2.4.10 Skenario Transaksi Pembayaran Sukses

Skenario ini menggambarkan alur transaksi pencatatan pembayaran yang sukses dilakukan dan informasinya disampaikan ke *client*. Diagram *sequence* untuk menggambarkan skenario ini seperti terlihat pada gambar 2.18 :



Gambar 2.18: Diagram *Sequence* Pencatatan Transaksi Pembayaran Yang Sukses

Skenario ini diawali dari pemanggilan *method* prosesPembayaran milik kelas SpptRestController, karena *method* ini lah yang menjadi tempat masuk seluruh proses dari fitur yang ditawarkan oleh *server web services* ini.

Secara berurut, *method* prosesPembayaran milik kelas SpptRestController, akan memanggil *method* prosesPembayaran milik kelas PembayaranServicesImpl, yang didalam *method* ini akan memanggil *method* prosesPembayaran milik kelas StoreProceduresDaoImpl.

Di dalam *method* prosesPembayaran milik kelas StoreProceduresDaoImpl, proses pemanggilan *method* executeUpdate milik kelas CallableStatement di-

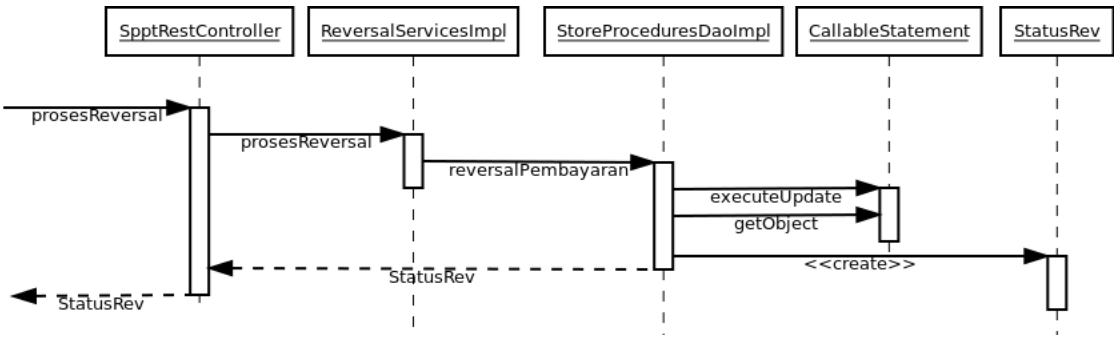
lakukan untuk mengeksekusi *store procedures* pada basis data, kemudian mengambil hasil nilainya dengan memanggil *method* getObject.

Selanjutnya, dari nilai-nilai yang dikembalikan oleh *store procedures* basis data, nilai-nilai tersebut ditampung pada kelas PembayaranSppt, yang kemudian dimasukkan ke dalam kelas StatusTrx untuk dikembalikan ke kelas SpptRestController. Kemudian kelas SpptRestController mengirimkan kelas StatusTrx sebagai respon dari *request* yang dikirimkan oleh *client*.

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

Skenario ini menggambarkan proses permintaan *reversal* yang gagal karena data yang diminta untuk dilakukan proses *reversal* tidak ada dalam basis data.

Diagram *sequence* dari skenario ini seperti terlihat pada gambar 2.19 :



Gambar 2.19: Diagram *Sequence Reversal* Yang Gagal Karena Data Tidak Ada Dalam Basis Data

Proses ini berawal dari *request* yang disampaikan oleh *client* yang kemudian diterima oleh *method* prosesReversal milik kelas SpptRestController.

Di dalam *method* prosesReversal milik kelas SpptRestController akan memanggil *method* prosesReversal milik kelas ReversalServicesImpl, yang di dalam *method*

ini memanggil *method* reversalPembayaran milik kelas StoreProceduresDaoImpl.

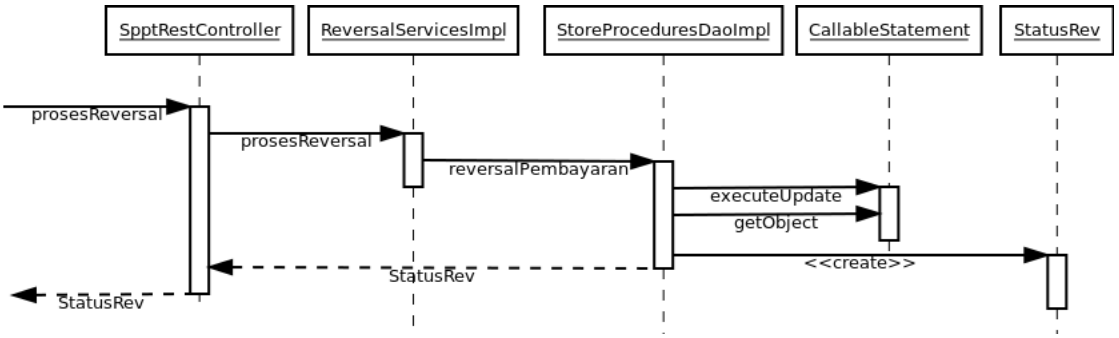
Pada *method* reversalPembayaran milik kelas StoreProceduresDaoImpl ini, akan mengeksekusi *store procedure* milik basis data dengan memanggil *method* executeUpdate milik kelas CallableStatement, dan mengambil nilai hasil eksekusinya melalui *method* getObject.

Hasil dari basis data yang kosong inilah menjadi dasar respon bahwa data yang diminta untuk dilakukan proses *reversal* tidak ada dalam basis data, kemudian *method* reversalPembayaran akan membuat instan dari kelas StatusRev dan mengembalikannya ke kelas SpptRestController, sehingga *server* akan memberikan respon ke *client* berupa kelas StatusRev ini.

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

Skenario ini akan menjelaskan bagaimana proses *request reversal* yang gagal karena adanya pembayaran ganda yang tercatat dalam basis data. Diagram *sequence* untuk skenario ini sama persis dengan kondisi skenario proses *reversal* yang gagal karena data yang diminta tidak ada. Ini karena baik respon terhadap pencatatan pembayaran ganda dan tidak ada data berasal dari *store procedure* milik basis data, sehingga diagram *sequence* akan nampak sama, tetapi respon datanya berbeda.

Diagram *sequence* untuk skenario ini seperti pada gambar 2.20 :



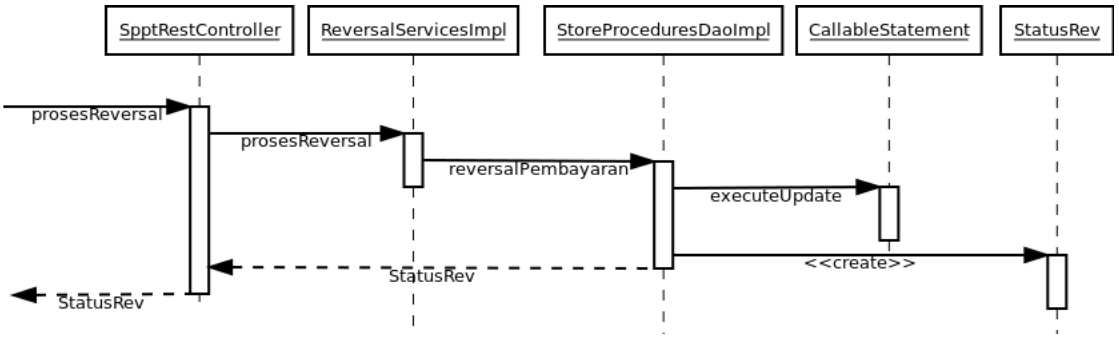
Gambar 2.20: Diagram *Sequence* Untuk Proses *Reversal* Yang Gagal Karena Data Pembayaran Tercatat Ganda

Proses ini berawal dari pemanggilan *method* prosesReversal milik kelas SpptRestController, dan alur yang sama seperti skenario proses *reversal* yang gagal karena data yang diminta tidak ada, yang membedakan adalah pada saat pemanggilan *method* getObject milik kelas CallableStatement, respon yang diberikan oleh *store procedure* berbeda dari skenario sebelumnya.

Sebagai respon dari *server* untuk *client*, maka dibentuklah instan dari kelas StatusRev untuk membungkus informasi status proses *reversal* yang terjadi.

2.4.13 Skenario *Reversal* Gagal Karena Kesalahan Server

Skenario ini menjelaskan alur proses *reversal* yang gagal karena ada kesalahan komunikasi antara *server* Rest dengan *server* basis data. Diagram *sequence* dari skenario ini seperti terlihat pada gambar 2.21 :

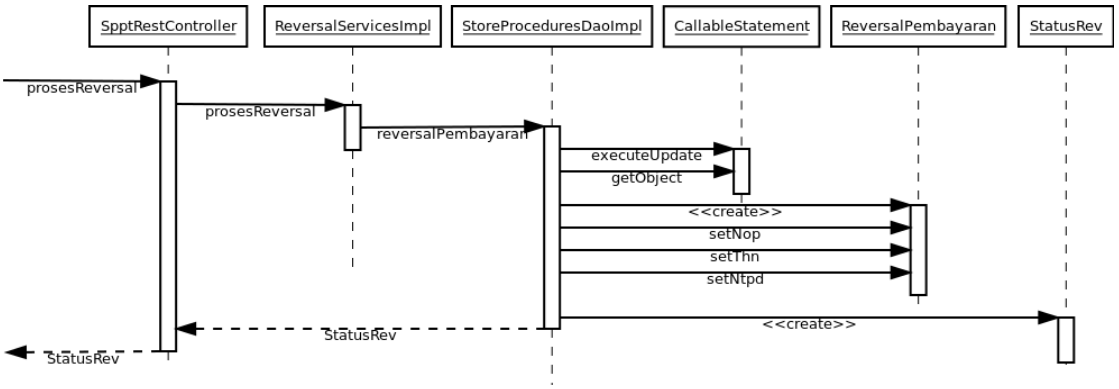


Gambar 2.21: Diagram *Sequence* Dari Proses *Reversal* Yang Gagal Karena Kesalahan Komunikasi Dengan Basis Data

Diagram ini pun terlihat mirip dengan skenario-skenario *reversal* yang gagal sebelumnya, hanya perbedaannya pada saat *method* reversalPembayaran milik kelas StoreProceduresDaoImpl memanggil *method* executeUpdate milik kelas *CallableStatement* ada kesalahan yang tidak diharapkan, sehingga proses keluar melalui *exception*, yang akhirnya *method* reversalPembayaran milik kelas StoreProceduresDaoImpl membuat instan kelas StatusRev, dan menjadi nilai kembalian untuk kelas SpptRestController yang selanjutnya menjadi respon *server* terhadap *request* dari *client*.

2.4.14 Skenario *Reversal* Sukses

Skenario ini akan menjelaskan proses *reversal* yang berhasil. Diagram *sequence* untuk menggambarkan skenario ini adalah seperti pada gambar 2.22 :



Gambar 2.22: Diagram *Sequence* Untuk Fungsi *Reversal*

Seperti beberapa *request* sebelumnya, *request reversal* yang sukses ini pun akan dimulai dari kelas SpptRestController pada *method* prosesReversal.

Pada *method* prosesReversal kelas SpptRestController akan memanggil *method* yang sama namanya, yaitu *method* prosesReversal tetapi milik kelas ReversalServicesImpl.

Dari *method* prosesReversal milik kelas ReversalServicesImpl ini, kemudian akan memanggil *method* reversalPembayaran pada kelas StoreProceduresDaoImpl yang didalamnya memuat koneksi ke basis data kemudian memanggil *store procedure* dengan menggunakan *method* executeUpdate milik kelas CallableStatement, dan mengambil nilai balikkan dari basis data dengan *method* getObject milik kelas CallableStatement juga yang hasilnya ditampung pada kelas ReversalPembayaran.

Pada langkah terakhir, *method* reversalPembayaran kelas StoreProceduresImpl ini akan membungkus kelas ReversalPembayaran di dalam kelas StatusRev yang dikembalikan ke kelas SpptRestController agar dijadikan bahan respon terhadap *request* yang dilakukan oleh *client*.

BAB 3

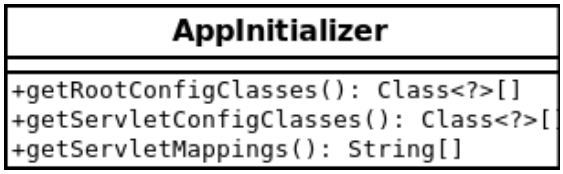
DESKRIPSI SUB SISTEM

Karena sistem yang akan dibuat menggunakan desain berorientasi objek, maka penjelasan lebih detail dari sub-sistem ini akan lebih mudah bila kita melihat diagram *class* yang telah dijelaskan dan digambarkan pada bagian sebelumnya, namun kali ini akan dijelaskan lebih detail.

3.1 Kelas AppInitializer

Kelas ini menjadi kelas pembuka bagi sistem aplikasi yang menggunakan *framework* Spring karena kelas ini mengimplementasikan *interface* dari *framework* Spring yang bernama AbstractAnnotationConfigDispatcherServletInitializer.

Diagram dari kelas AppInitializer ini seperti ditunjukkan pada gambar 3.1 :



Gambar 3.1: Diagram Kelas AppInitializer

Karena kelas ini adalah implementasi dari *interface* AbstractAnnotation-

ConfigDispatcherServletInitializer, maka harus mengimplementasikan pula ketiga *method* yang ada pada *interface* tersebut, yaitu *method* getRootConfigClasses, *method* getServletConfigClasses, dan *method* getServletMappings.

3.2 Kelas AppConfig

Kelas ini merupakan kelas konfigurasi yang digunakan *framework* Spring untuk melakukan konfigurasi terhadap sistem aplikasi yang akan dijalankan. Bukan karena mengimplementasikan kelas atau *interface* yang dibawa oleh Spring, melainkan karena dideklarasikan sebagai kelas yang menangani konfigurasi sistem aplikasi di kelas AppInitializer.

Pada kelas ini, karena menggunakan fitur MVC (*Model-View-Controller*) dari Spring, maka akan ada beberapa kelas yang ditujukan untuk fungsinya masing-masing, dikelas inilah nantinya kelas-kelas tersebut dideklarasikan, pada kelas ini pula disebutkan kelas yang bertanggung jawab untuk melakukan konfigurasi-konfigurasi yang lain termasuk konfigurasi komunikasi dengan sistem basis data.

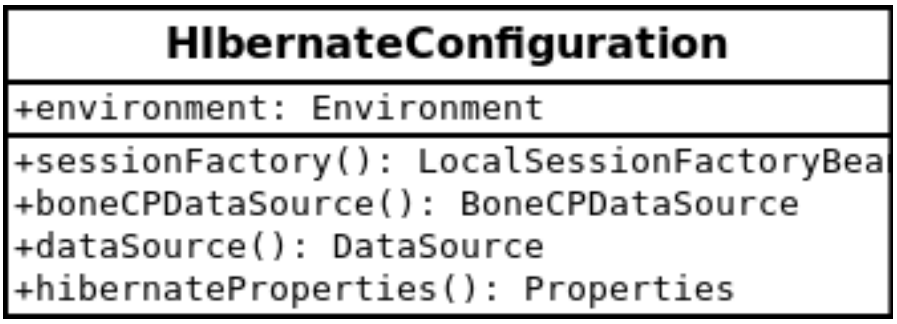
Diagram dari kelas AppConfig ini seperti ditampilkan pada gambar 3.2 :

Gambar 3.2: Diagram Kelas AppConfig

Kelas ini memang terlihat kosong, karena konfigurasi-konfigurasi yang dilakukan nantinya akan menggunakan fitur *annotation* milik Java agar memudahkan pembacaan kode dan menyederhanakan konfigurasi.

3.3 Kelas HibernateConfiguration

Kelas ini bertugas melakukan konfigurasi komunikasi dengan sistem basis data. Diagram dari kelas HibernateConfiguration ini seperti terlihat pada gambar 3.3 :



Gambar 3.3: Diagram Kelas HibernateConfiguration

Kelas ini terdiri dari 1 (satu) properti dan 4 (empat) *method*. Propertinya diberi nama environment, karena properti ini akan bertugas menampung nilai-nilai konfigurasi standar yang berada pada *file* yang terpisah.

Method yang pertama adalah *method* sessionFactory. *Method* sessionFactory ini akan bertugas mengembalikan nilai sebuah kelas LocalSessionFactoryBean dimana kelas ini yang menjadi sesi dari tiap *user* untuk melakukan koneksi ke basis data nantinya. Untuk menyediakan akses data yang cepat dan optimal, *method* ini akan menggunakan *connection pool* milik BoneCP.

Method berikutnya adalah boneCPDataSource, *method* ini akan menyediakan konfigurasi untuk melakukan koneksi ke basis data sebagai sumber data, yang dikembalikan dalam bentuk kelas BoneCPDataSource. *Method* ini menjadi sumber untuk pengambilan data apapun pada basis data dengan menggunakan *connection pool* BoneCP.

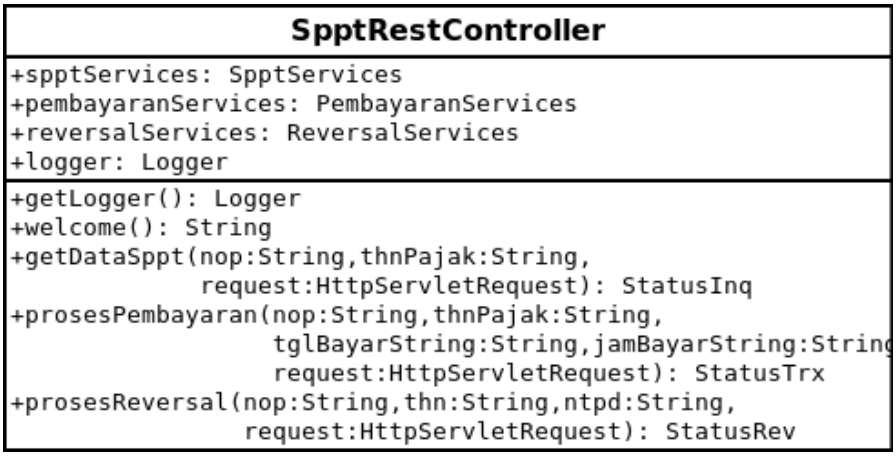
Method dataSource adalah *method* standar yang menggunakan *connection pool* yang dibawa oleh Hibernate yang digunakan untuk melakukan pemeriksaan awal

koneksi ke basis data, apabila menggunakan *method* ini berhasil, maka tidak akan menjadi masalah apabila dilakukan menggunakan *connection pool* yang lain.

Method yang terakhir adalah hibernateProperties, *method* ini melakukan konfigurasi terhadap Hibernate yang nantinya digunakan pada *method* sessionFactory.

3.4 Kelas SpptRestController

Kelas ini yang menjadi gerbang pertama yang menentukan kemana sebuah *request* akan di respon. Diagram kelas dari SpptRestController ini akan terlihat seperti pada gambar 3.4 :



Gambar 3.4: Diagram Kelas SpptRestController

Kelas ini memiliki 4 (empat) properti dan 5 (lima) *method*. Penjelasan dan fungsi masing-masing properti dan *method* tersebut yaitu sebagai berikut :

- Properti
  - Properti spptServices

    Properti ini yang akan mengatur layanan yang menangani permintaan / *request inquiry* dari *client*.

- Properti pembayaranServices

  Properti pembayaranServices ini akan mengatur layanan yang menangani permintaan / *request* pencatatan pembayaran dari *client*.

- Properti reversalServices

  Properti reversalServices ini akan mengatur layanan yang menangani permintaan / *request reversal* dari *client* karena ada kesalahan pencatatan pembayaran sebelumnya.

- Properti logger

  Properti logger digunakan untuk melakukan pencatatan aktifitas sistem aplikasi ke dalam sebuah *file* pada saat sistem aplikasi berjalan dan digunakan. Kondisi ini diperlukan agar memudahkan diagnosa sistem apabila terjadi anomali-anomali proses yang tidak diinginkan.

- *Method*

  - *Method* getLogger

    Method getLogger ini menyediakan akses ke properti logger agar dapat mencatat kejadian saat *runtime* dari kelas mana pun pada sistem aplikasi.

  - *Method* welcome

    Method welcome ini adalah halaman muka dari layanan Rest, jadi setiap *request* yang dilayangkan ke *root slash* ("/"), maka akan direspon oleh *method* welcome ini.

  - *Method* getDataSppt

    Method getDataSppt difungsikan sebagai tempat masuknya *request inquiry* data tagihan SPPT PBB-P2, nantinya *method* ini akan mengem-

balikan respon dalam bentuk kelas StatusInq yang telah diubah dalam format JSON.

– *Method* prosesPembayaran

Method prosesPembayaran ini akan bertugas menangani *request* pencatatan pembayaran dari *client*. *Method* ini akan mengembalikan kelas StatusTrx yang tentunya telah diubah dalam format JSON.

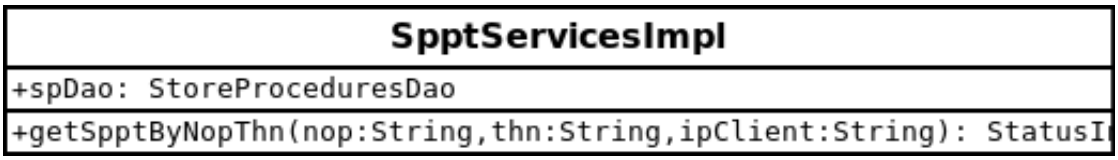
– *Method* prosesReversal

Method prosesReversal ini seperti *method* getDataSppt dan *method* prosesPembayaran yang menangani sebuah *request* dari *client*, namun *method* prosesReversal ini akan menangani *request reversal* dari pembayaran yang telah tercatat dalam basis data.

Penyebab proses *reversal* ini karena adanya kesalahan pencatatan pembayaran yang sebelumnya terjadi, misalkan pada saat *client* melakukan *request* pencatatan pembayaran, sebelum *client* mendapatkan respon dari *server* koneksi tiba-tiba terputus dan *client* tidak pernah mendapatkan respon atas *request*-nya, maka untuk memastikan adalah melakukan *request reversal* dan melakukan *request* pencatatan pembayaran kembali.

3.5 Kelas SpptServicesImpl

Kelas SpptServicesImpl akan melayani *request inquiry* dan melanjutkannya ke kelas-kelas yang menangani basis data. Diagram kelas dari SpptServicesImpl ini adalah seperti pada gambar 3.5



Gambar 3.5: Diagram Kelas SpptServicesImpl

Kelas SpptServicesImpl ini memiliki 1 (satu) properti dan 1 (satu) *method*. Propertinya bernama spDao, untuk menampung kelas StoreProceduresDao yang menangani semua hal mengenai eksekusi *store procedure* pada basis data.

Method dari kelas ini bernama getSpptByNopThn yang berfungsi melakukan pemanggilan terhadap kelas yang bertugas menghubungi basis data yang pencarian datanya berdasarkan Nomor Objek Pajak (NOP) dan tahun pajak.

3.6 Kelas PembayaranServicesImpl

Kelas PembayaranServicesImpl bertugas melayani permintaan terhadap pencatatan pembayaran yang posisinya berada di tengah dan menjadi penghubung antara *interface* dengan kelas-kelas *data access*. Diagram kelas PembayaranServicesImpl ini seperti pada gambar 3.6 :

Gambar 3.6: Diagram Kelas PembayaranServicesImpl

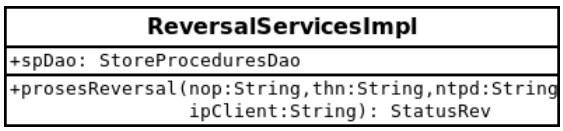
Kelas ini memiliki 1 (satu) properti dan 1 (satu) *method*, properti yang dimiliki bernama spDao, ini adalah instan dari kelas atau *interface* StoreProceduresDao yang bertugas melakukan komunikasi dengan sistem aplikasi basis data.

Sedangkan *method* prosesPembayaran adalah *method* yang bertugas sebagai penghubung antara *interface* dalam hal ini kelas SpptRestController, dengan kelas yang berhubungan dan berkomunikasi langsung dengan basis data untuk melakukan eksekusi pencatatan pembayaran.

3.7 Kelas ReversalServicesImpl

Seperti kelas PembayaranServicesImpl dan SpptServicesImpl, kelas ReversalServicesImpl bertugas menjadi penghubung antara *interface* yang menangani proses *reversal* dengan kelas-kelas yang berhubungan dengan basis data.

Diagram kelas ReversalServicesImpl ini seperti terlihat pada gambar 3.7 :



Gambar 3.7: Diagram Kelas ReversalServicesImpl

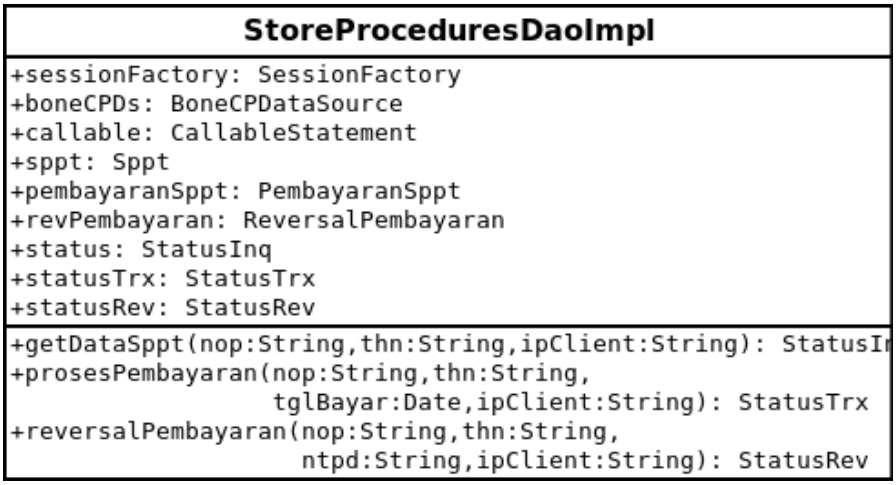
Seperti terlihat pada diagram bahwa kelas ReversalServicesImpl ini memiliki sebuah properti dan sebuah *method*.

Properti yang dimiliki kelas ReversalServicesImpl ini bernama spDao yang merupakan instan dari kelas StoreProceduresDao, dengan *method* bernama prosesReversal, *method* ini nantinya akan berkomunikasi dengan kelas StoreProceduresDao melalui instan kelas spDao yang menjadi penghubung dengan sistem aplikasi basis data.

3.8 Kelas StoreProceduresDaoImpl

Kelas StoreProceduresDaoImpl ini yang mengimplementasikan *interface* StoreProceduresDao yang fungsinya melakukan komunikasi dengan sistem basis data. Di-

agram kelas StoreProceduresDaoImpl ini seperti terlihat pada gambar 3.8 :



Gambar 3.8: Diagram Kelas StoreProceduresDaoImpl

Kelas ini memiliki beberapa properti dan 3 (tiga) buah *method*. Properti yang pertama bernama sessionFactory, ini adalah instan dari kelas SessionFactory yang bertugas membuka sesi komunikasi dengan sistem basis data.

Properti yang kedua adalah boneCPDs, yang merupakan instan dari kelas BoneCPDataSource. Properti ini akan melaksanakan tugasnya dengan *framework* Hibernate menggunakan *connection pool* dari BoneCP untuk hasil komunikasi yang lebih optimal.

Properti yang ketiga adalah callable yang merupakan instan dari kelas CallableStatement, properti ini akan menyediakan perangkat komunikasi yang mampu melakukan eksekusi terhadap *store procedure* yang berada pada basis data.

Properti selanjutnya adalah sppt, pembayaranSppt, dan revPembayaran yang secara berurutan adalah instan dari kelas Sppt, PembayaranSppt, dan ReversalPembayaran. Ketiga properti ini memiliki fungsi yang sama, yaitu menampung nilai yang dikembalikan saat melakukan eksekusi *store procedure* dari sistem basis data.

Properti berikutnya yaitu status, statusTrx, dan statusRev, yang secara berurutan merupakan instan dari kelas StatusInq, StatusTrx, dan StatusRev. Fungsi dari ketiga properti ini yaitu sebagai pembungkus, atau penjelas informasi terhadap respon dari *request* yang dikirimkan oleh *client*.

Untuk *method* yang pertama adalah *method* getDataSppt, yang fungsinya melakukan komunikasi dengan sistem basis data untuk memberikan respon terhadap *request inquiry* dari *client*.

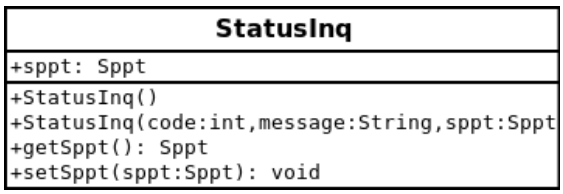
Method yang kedua adalah *method* prosesPembayaran, yang fungsinya melakukan komunikasi dengan sistem basis data untuk memberikan respon terhadap *request* pencatatan pembayaran di *client*.

Sedangkan *method* reversalPembayaran berfungsi melakukan komunikasi dengan sistem basis data untuk memberikan respon terhadap *request reversal* pembayaran dari *client*.

3.9 Kelas StatusInq

Kelas StatusInq ini digunakan untuk membungkus data respon sebuah *inquiry* menjadi informasi yang lebih jelas karena ada penjelasan status respon yang diberikan. Misalkan, data *inquiry* tidak muncul, maka kelas ini akan memberikan informasi lain apakah nomor objek pajak yang diminta ada atau tidak, atau kondisi nomor objek pajak tidak muncul karena telah terbayarkan.

Diagram kelas StatusInq ini seperti terlihat pada gambar 3.9 :



Gambar 3.9: Diagram Kelas StatusInq

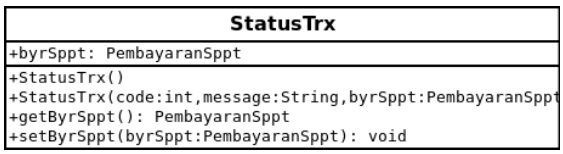
Kelas ini hanya memiliki 1 (satu) properti, 2 (dua) konstruktor, dan 2 (dua) *method*.

Properti pada kelas ini diberi nama sppt yang merupakan instan dari kelas Sppt. Seperti dijelaskan pada bagian sebelumnya, kelas Sppt ini digunakan untuk menampung nilai yang dikembalikan oleh sistem basis data pada saat pemanggilan *store procedure inquiry*.

Kontruktor disediakan 2 (dua), untuk menjadikan kelas ini fleksibel dalam implementasi instan. Sedangkan 2 (dua) *method* yang disediakan sebetulnya untuk melakukan akses *set* dan *get* terhadap properti sppt.

3.10 Kelas StatusTrx

Kelas StatusTrx ini berfungsi membungkus informasi pencatatan transaksi pembayaran agar respon kepada *client* lebih jelas. Diagram kelas dari kelas StatusTrx ini seperti terlihat pada gambar 3.10 :



Gambar 3.10: Diagram Kelas StatusTrx

Seperti kelas StatusInq, kelas ini pun memiliki sebuah properti, 2 (dua) buah kontruktor, dan 2 (dua) buah *method*.

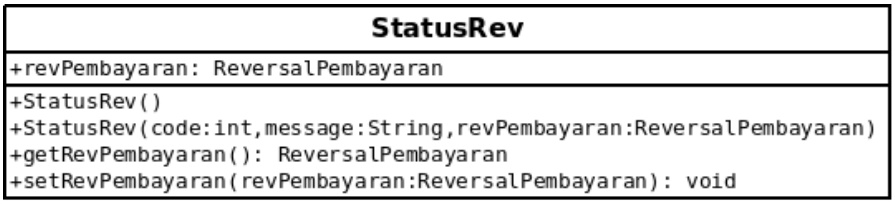
Properti pada kelas ini bernama byrSppt yang merupakan instan dari kelas PembayaranSppt, yang fungsinya adalah menampung hasil respon dari sistem basis data atas eksekusi *store procedure* untuk pencatatan pembayaran.

Fungsi dari 2 (dua) kontruktor yang ada pada kelas ini pun ditujukan agar pembentukan instan agar lebih fleksibel. Kedua *method* pun dimaksudkan untuk

memberikan akses pada properti dari kelas ini.

3.11 Kelas StatusRev

Kelas StatusRev ini berfungsi untuk membungkus informasi *reversal* agar respon terhadap *request client* lebih jelas. Diagram kelas StatusRev ini dapat dilihat pada gambar 3.11 :



Gambar 3.11: Diagram Kelas StatusRev

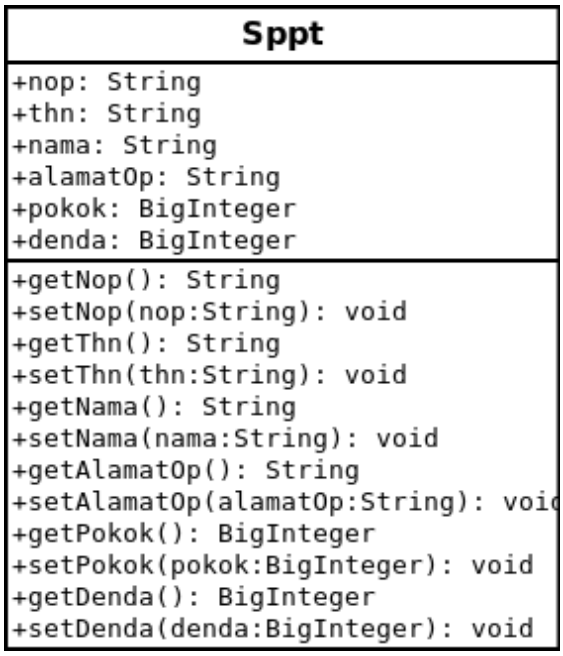
Seperti kelas StatusInq dan StatusTrx, kelas ini pun terdiri dari sebuah properti, 2 (dua) buah konstruktor, dan 2 (dua) buah *method*.

Properti kelas ini bernama revPembayaran yang merupakan instan dari kelas ReversalPembayaran, yang fungsinya menampung hasil respon dari basis data terhadap eksekusi *store procedure reversal* pembayaran.

Kedua konstruktor yang disediakan untuk menjadikan pembuatan instan kelas ini lebih fleksibel, sedangkan dua *method* yang disediakan merupakan fasilitas untuk melakukan akses terhadap properti kelas ini.

3.12 Kelas Sppt

Kelas Sppt ini adalah kelas yang menampung hasil respon dari pemanggilan atau eksekusi *store procedure inquiry* yang ada pada basis data. Diagram kelas ini seperti terlihat pada gambar 3.12 :



Gambar 3.12: Diagram Kelas Sppt

Kelas Sppt ini terdiri dari 6 (enam) properti dan 12 (dua belas) *method*. Enam properti ini adalah nilai-nilai yang dikembalikan dari hasil eksekusi *store procedure* pada basis data. Properti pada kelas ini terdiri dari :

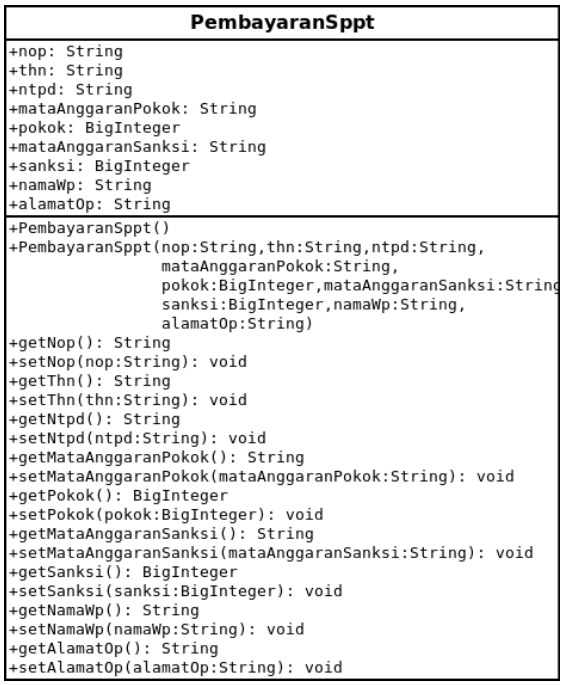
- nop. Properti ini untuk menyimpan Nomor Objek Pajak, sebagai identitas kunci, tiap objek pajak akan memiliki NOP yang berbeda-beda.
- thn. Properti ini untuk menampung Tahun Pajak dimana objek pajak dimunculkan tagihan pajaknya.
- nama. Properti ini akan menampung nama dari wajib pajak untuk nomor objek pajak tersebut pada properti nop.
- alamatOp. Properti ini untuk menampung alamat dari objek pajak, yang nantinya bisa digunakan sebagai bahan verifikasi bahwa objek tersebut adalah benar yang akan dicari tahu informasinya.

- pokok. Properti ini untuk menampung besarnya tagihan pokok atas nomor objek pajak yang tersebut pada properti nop.
- denda. Properti ini akan menampung besarnya denda administrasi yang ditetapkan apabila ada keterlambatan pembayaran pajak yang telah melewati tanggal jatuh temponya.

Seluruh *method* yang disediakan pada kelas ini hanya bertujuan untuk melakukan akses terhadap properti pada kelas ini.

3.13 Kelas PembayaranSppt

Kelas PembayaranSppt berfungsi untuk menampung informasi dari respon pencatatan pembayaran SPPT yang terjadi. Diagram kelas dari PembayaranSppt ini seperti terlihat pada gambar 3.13 :



Gambar 3.13: Diagram Kelas PembayaranSppt

Kelas ini memiliki 9 (sembilan) properti, 2 (dua) kontruktor, dan 18 (delapan belas) *method*. Properti dari kelas ini terdiri dari :

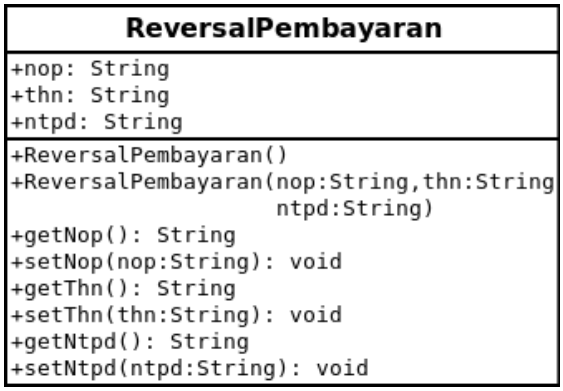
- nop. Properti nop ini digunakan untuk menampung Nomor Objek Pajak.
- thn. Properti thn ini digunakan untuk menampung Tahun Pajak dimana NOP dikenakan Pajak Bumi dan Bangunan.
- ntpd. Properti ntpd ini digunakan untuk menampung Nomor Transaksi Penerimaan Daerah, yang menjadi identitas bahwa pembayaran atas NOP dan Tahun pajak yang diinginkan telah tercatat dalam basis data SISMIOP.
- mataAnggaranPokok. Properti mataAnggaranPokok ini akan menyimpan kode mata anggaran untuk realisasi penerimaan pokok pajak untuk jenis Pajak Bumi dan Bangunan Perdesaan dan Perkotaan.
- pokok. Properti pokok ini akan menampung besarnya pokok pajak yang dibayarkan oleh wajib pajak.
- mataAnggaranSanksi. Properti mataAnggaranSanksi ini akan menampung kode mata anggaran untuk realisasi sanksi administrasi berupa denda dari keterlambatan pembayaran Pajak Bumi dan Bangunan Perdesaan dan Perkotaan.
- sanksi. Properti sanksi ini akan menampung besarnya denda yang dibayarkan oleh wajib pajak karena keterlambatan pembayaran pokok pajak untuk jenis PBB-P2.
- namaWp. Properti namaWp ini akan menampung Nama Wajib Pajak yang tercantum dalam SPPT PBB-P2 pada saat terjadinya transaksi pembayaran.
- alamatOp. Properti alamatOp ini akan menampung Alamat Objek Pajak yang tercantum dalam SPPT-P2 pada saat terjadinya transaksi pembayaran.

Dua buah kontruktor yang disediakan seperti pada kelas-kelas sebelumnya, ditujukan untuk fleksibilitas pembentukan instan dari kelas PembayaranSppt ini.

Untuk 18 (delapan belas) *method* yang disediakan, ditujukan untuk keperluan akses properti yang dimiliki oleh kelas PembayaranSppt ini.

3.14 Kelas ReversalPembayaran

Kelas ReversalPembayaran ini berfungsi menampung data-data transaksi *reversal* pembayaran PBB-P2. Diagram kelas ReversalPembayaran ini seperti terlihat pada gambar 3.14 :



Gambar 3.14: Diagram Kelas ReversalPembayaran

Kelas ini memiliki 3 (tiga) buah properti, 2 (dua) konstruktor, dan 6 (enam) buah *method*.

Properti dari kelas ReversalPembayaran ini terdiri dari :

- nop. Properti nop ini berfungsi untuk menampung Nomor Objek Pajak sebagai identitas objek pajak yang akan dilakukan *reversal* pembayarannya.
- thn. Properti thn ini berfungsi untuk menampung tahun pajak yang akan dilakukan *reversal* pembayarannya.

- ntpd. Properti ntpd ini berfungsi untuk menampung Nomor Transaksi Penerimaan Daerah yang akan dilakukan *reversal* pembayarannya.

Dua konstruktor seperti pada kelas-kelas sebelumnya, ditujukan agar pembentukan instan dari kelas ReversalPembayaran ini menjadi lebih fleksibel. Begitu juga dengan 6 (enam) buah *method* yang disediakan kelas ini, difungsikan untuk melakukan akses pada kelas ReversalPembayaran.

Selain kelas-kelas pembentuk aplikasi, di tingkat sistem basis data dibentuk *store procedure*. Daftar *store procedure* yang berada pada basis data adalah sebagai berikut :

- SPPT_TERHUTANG

  Store procedure SPPT_TERHUTANG ini nantinya akan mengambil data dari tabel SPPT dan mengembalikan nilainya dalam kolom NOP (Nomor Objek Pajak), THN (Tahun Pajak), NAMA (Nama Wajib Pajak), ALAMAT_OP (Alamat Objek Pajak), POKOK (Besarnya PBB-P2 terhutang), dan DENDA (Besarnya denda administrasi bila ada).

- PROSES_PEMBAYARAN.

  Store procedure PROSES_PEMBAYARAN ini nantinya akan mengambil data dari tabel SPPT, DAT_OP_BUMI, REF_KELURAHAN, dan REF_KECAMATAN. Keempat tabel tersebut sebagai bahan untuk diformulisikan agar dapat memenuhi pencatatan pada tabel PEMBAYARAN_SPPT yang merupakan tempat mencatatkan pembayaran PBB-P2 yang telah terjadi.

  Hasil dari pencatatan pada tabel PEMBAYARAN_SPPT kemudian dijadikan respon pencatatan ke aplikasi yang melakukan eksekusi *store procedure* ini. Keseluruhan transaksi pencatatan tabel PEMBAYARAN_SPPT dan responnya ini kemudian dicatatkan kondisinya pada

tabel LOG_TRX_PEMBAYARAN yang memuat NTPD (Nomor Transaksi Penerimaan Daerah).

- REVERSAL_PEMBAYARAN.

  Store procedure REVERSAL_PEMBAYARAN ini nantinya akan mengambil data dari tabel LOG_TRX_PEMBAYARAN, bila ada data pada tabel LOG_TRX_PEMBAYARAN, maka data NOP (Nomor Objek Pajak) untuk Tahun Pajak tersebut pada *request* akan dihapus dari tabel PEMBAYARAN_SPPT, dan mengubah isi kolom STATUS_PEMBAYARAN_SPPT pada tabel SPPT menjadi 0 (karakter nol).

  Keseluruhan proses transaksi *reversal* ini dicatat pada tabel LOG_REVERSAL.

BAB 4

PERTIMBANGAN KHUSUS KINERJA SISTEM

Sebagai bahan pertimbangan khusus agar sistem ini menjadi lebih sempurna, maka diperlukan kondisi jaringan yang aman.

Maksudnya adalah, karena bentuk *web services* REST dapat diakses oleh siapa pun dan dari mana pun, maka diperlukan filter keamanan pada sisi jaringan, artinya, hanya ada beberapa alamat IP saja yang dapat melakukan akses ke *server web services* agar transaksi berjalan lebih aman.

Kondisi jalur jaringan internet pun diperlukan lebih dari 1 (satu) jalur. Hal ini untuk melakukan penjagaan jalur komunikasi apabila ada 1 *line* terputus karena faktor diluar kewenangan administrator pada DPPK maka *line* yang lain akan melakukan *backup* koneksi.

BAB 5

HASIL PEMODELAN

Hasil pemodelan dari sistem aplikasi yang akan dibangun, secara internal sebetulnya sudah dibahas pada bagian Arsitektur Sistem dan diperjelas dengan penjelasan pada bagian Deskripsi Sub Sistem.

Secara tampilan tatap muka (*interface*) tidak akan menghasilkan apa-apa selain model dari format *request* yang akan menjadi 3 (tiga) bagian berikut :

1. `pospbb/sppt/{nop}/{thn}`

   Request ini digunakan untuk *inquiry* data tagihan PBB-P2, dimana `nop` nantinya digantikan dengan nomor objek pajak PBB-P2 tanpa tanda baca, dan `thn` adalah tahun pajak dimana NOP tersebut akan dilihat informasinya.

2. `pospbb/bayar/{nop}/{thn}/{tglBayar}/{jamBayar}`

   Request ini digunakan untuk melakukan pencatatan pembayaran, dimana `nop` adalah Nomor Objek Pajak yang dibayarkan, `thn` adalah tahun pajak yang dibayarkan, `tglBayar` adalah tanggal terjadinya pembayaran dalam format DDMMYYYY dimana DD adalah 2 (dua) digit tanggal dalam satu bulan, MM adalah 2 (dua) digit bulan, dan YYYY adalah tahun. `jamBayar` adalah jam dibayarkannya PBB-P2 dengan format HH24MI, dengan HH24

dimaksudkan adalah 2 (dua) digit jam dengan format 24 jam, dan MI adalah menitnya.

3. `pospbb/reversal/{nop}/{thn}/{ntpd}`

   Request ini adalah *request* untuk melakukan *reversal* data pembayaran, dimana `nop` adalah Nomor Objek Pajak yang akan dilakukan *reversal*, `thn` adalah Tahun pajak untuk NOP yang akan dilakukan *reversal*, sedangkan `ntpd` adalah Nomor Transaksi Penerimaan Daerah sebagai identitas pencatatan pembayaran yang telah terjadi.

Ini adalah format standar yang nantinya digunakan untuk berkomunikasi dengan *client* dalam hal ini pihak Bank sebagai tempat pembayaran dalam hal pencatatan transaksi pembayaran.

BAB 6

BIAYA DAN JADWAL

Biaya yang dibutuhkan untuk membangun sistem aplikasi ini secara pembangunan sistem tidak ada, tetapi ada faktor pendukung agar sistem dapat berkomunikasi dengan *client* yaitu :

- Adanya VPN Server yang dapat mengamankan lalu lintas data melalui jaringan internet yang sifatnya publik dengan spesifikasi RAM ... dan prosesor ... yang diharapkan dapat melayani permintaan VPN *Client* yang semakin bertambah. Rentang harganya akan terlihat relatif dari kisaran .. sampai dengan ...
- Adanya jaringan internet, dalam hal ini *bandwidth* yang mampu menangani transaksi ratusan, bahkan ribuan dalam waktu yang hampir bersamaan. Kondisi akses jaringan internet ini diusahakan yang berjenis *dedicated* dimana perbandingan unduh dan unggah akan berada pada 1:1. Kisaran harga untuk layanan ini pun variatif dari harga ... sampai dengan ... yang biasanya bergantung pada besarnya *bandwidth* yang dibutuhkan.