KODE/RUMPUN ILMU: 458/TEKNIK INFORMATIKA

A

LAPORAN PENELITIAN INTERNAL KELOMPOK MONODISIPLIN

JUDUL PENELITIAN

PENERAPAN *BATCH PROCESSING* DALAM SISTEM VALIDASI JARAK DAN HARGA UNTUK LAYANAN GRATIS ONGKOS KIRIM PADA *E-COMMERCE*

Ketua
Stephanie Pamela Adithama, ST., MT.
(NPP.01.14.875 /NIDN: 0527048801)

Anggota
Eduard Rusdianto, ST, MT.
(NPP. 06.97.623 /NIDN: 0502107201)

UNIVERSITAS ATMA JAYA YOGYAKARTA
FAKULTAS TEKNOLOGI INDUSTRI
PROGRAM STUDI INFORMATIKA

YOGYAKARTA
APRIL 2020

HALAMAN PENGESAHAN LAPORAN PENELITIAN INTERNAL KELOMPOK			
1	Judul Penelitian	Penerapan *Batch Processing* dalam Sistem Validasi Jarak dan Harga untuk Layanan Gratis Ongkos Kirim pada *E-Commerce*	
2	Kategori Penelitian	A. Penelitian diorientasikan pada penerbitan artikel jurnal ilmiah	
3	Tema Penelitian Universitas	E. Adaptif dengan Kebutuhan Global	
4	Topik Penelitian Unit	*Information Storage and Retrieval*	
IDENTITAS PENELITI			
5	Nama Ketua Peneliti	Stephanie Pamela Adithama, S.T., M.T.	
	Jabatan/Golongan	Asisten Ahli / III-B	
	NPP/NIDN	01.14.875	0527048801
	Bidang Keahlian	Teknik Informatika	
	Unit/Fakultas/Jurusan	Unit	Jurusan/Program Studi
		Fakultas Teknologi Industri	PS. Teknik Informatika
	Alamat Rumah	Jl. Babarsari Kompleks Dirgantara 2 No. 4 Yogyakarta	
	No. Telp/Faks/Email Ketua	0274-487711	Email : stephanie.pamela@uajy.ac.id
6	Anggota Peneliti	Anggota-1	Anggota-2
	Nama Anggota Peneliti	Eduard Rusdianto, S.T., M.T.	
	Jabatan/Golongan	Lektor / III-C	
	NPP/NIDN	06.97.623 / 0502107201	
	Bidang Keahlian	Teknik Informatika	
	Unit/Fakultas/Jurusan	Fakultas Teknologi Industri/ Program Studi Teknik Informatika	
7	Lokasi Penelitian	Universitas Atma Jaya Yogyakarta	
	Waktu Penelitian	Agustus 2019 – Desember 2019	
8	Dana yang diusulkan	Dana UAJY	
	Jumlah Total	Rp 20.000.000,00	
	Terbilang	Dua Puluh Juta Rupiah	
9	Spesifikasi *Outcome* Penelitian	Sebuah *prototype* perangkat lunak dan paper penelitian yang dipublikasikan minimal di jurnal internasional	

Yogyakarta, April 2020

Mengetahui dan Menyetujui,
Ketua Program Studi Teknik Informatika,

Martinus Maslim, S.T., M.T.
NPP.01.13.847/NIDN.0512039002

Pengusul,

Stephanie Pamela Adithama, ST, MT.
NPP:01.14.875/NIDN:0527048801

Dekan Fakultas Teknologi Industri,

Dr. A. Teguh Siswantoro, M.Sc.
NPP. 09.93.464/NIDN.0521115901

Mengetahui dan Menyetujui,
Ketua LPPM,

Prof. Suyoto, M.Sc., Ph.D.
NPP: 09.00.686/NIDN: 0510086401

DAFTAR ISI

RINGKASAN

Layanan gratis ongkos kirim merupakan salah satu fitur yang ditawarkan oleh beberapa *e-commerce* dalam rangka promosi dan menarik konsumen untuk melakukan transaksi jual beli. Perusahaan *e-commerce* PT. XYZ menerapkan layanan tersebut untuk barang-barang yang memenuhi kriteria perusahaan. Setiap bulan perusahaan akan menerima tagihan transaksi pengiriman dari perusahaan pengiriman pihak ketiga, dalam bentuk berkas Excel. Divisi operasional akan melakukan proses validasi menggunakan perhitungan secara manual antara jarak dan harga yang sesungguhnya, dibandingkan dengan data tagihan dalam berkas Excel. Proses manual yang dilakukan ini memunculkan beberapa masalah seperti lamanya waktu pemrosesan dan kesalahan yang disebabkan oleh faktor manusia. Oleh sebab itu, dilakukan penelitian ini yang bertujuan untuk membangun sebuah sistem validasi, agar proses validasi ini dapat berjalan secara optimal.

Sistem akan menggunakan *batch processing* untuk memproses data yang besar tanpa interupsi. Sistem akan memanfaatkan *framework* yaitu Spring Batch. *Batch Processing* akan mengeksekusi beberapa proses dalam waktu bersamaan dan membagi ke dalam sejumlah potongan. Bahasa pemrograman yang digunakan adalah Java dengan *framework* yaitu Spring Boot. Sistem dapat menerima berkas Excel yang berisi sekumpulan data tagihan pengiriman dan kemudian mengolahnya, sehingga menghasilkan laporan. Google Maps API dilibatkan dalam sistem untuk mendapatkan jarak terhadap suatu titik *longitude* dan *latitude* yang diberikan.

Penelitian ini dapat membantu divisi operasional pada perusahaan *e-commerce* untuk membantu mempercepat proses validasi jarak dan harga. Jumlah rata-rata data yang mampu seorang karyawan kerjakan, per hari dengan waktu dua jam kalkulasi jarak dan harga adalah 500 data. Setelah sistem diimplementasikan, untuk 500 data, proses kalkukasi jarak dan harga membutuhkan waktu 120 detik. Melalui perbandingan terhadap proses validasi, sebelum dan sesudah sistem diimplementasikan, maka presentase untuk pengurangan waktu dalam proses validasi adalah 98.33%.

Kata Kunci : *batch processing*, *e-commerce*, *spring batch*, sistem validasi, layanan gratis ongkos kirim

BAB 1

PENDAHULUAN

Peningkatan pemanfaatan internet dan teknologi dalam beberapa waktu belakangan ini memberikan dampak pada beberapa aspek kehidupan. Salah satunya adalah dengan munculnya *e-commerce* pada model bisnis di Indonesia. Perdagangan elektronik atau *e-commerce* merupakan suatu proses yang mendukung pelanggan, menyediakan layanan dan komoditas, bagian dari informasi bisnis, mengelola transaksi bisnis dan mempertahankan ikatan antara penyedia, pelanggan, dan vendor dengan perangkat jaringan telekomunikasi (Nanehkaran, 2013). Kehadiran *e-commerce* mendapatkan sambutan baik dari masyarakat, menurut ketua umum Indonesian *E-Commerce Association* (iDEA) bahwa data sensus ekonomi 2016 dari Badan Pusat Statistik (BPS) menyebutkan, *e-commerce* Indonesia dalam sepuluh tahun terakhir tumbuh sekitar tujuh belas persen dengan total jumlah usaha *e-commerce* mencapai 26,2 juta unit. Dampak *e-commerce* pada model bisnis tradisional yaitu mempercepat generasi industri baru, mengurangi biaya pembeli, dan mempercepat siklus pengembangan produk baru (Liu, 2016).

Perusahaan PT. XYZ merupakan sebuah *e-commerce*, dengan jumlah pelanggan yang besar dan transaksi yang terus meningkat. Untuk melakukan pengiriman barang pada perusahaan berbasis jual-beli *online*, pada umumnya perusahaan akan menggunakan jasa dari *third party logistic*. Pelanggan akan dikenakan biaya ongkos kirim untuk setiap transaksi yang dilakukan. *Third party logistic* merupakan suatu perusahaan penyedia layanan logistik eksternal yang menawarkan satu atau beberapa kegiatan logistik berbasis kontrak atau kesepakatan (Yang, 2014).

Dalam rangka menambah nilai unggul di antara *e-commerce* lainnya, perusahaan menerapkan layanan gratis ongkos kirim terhadap transaksi yang telah memenuhi persyaratan. Biaya pengiriman tersebut akan ditanggung oleh perusahaan, pelanggan tidak perlu membayar ongkos kirimnya. *Third party logistic* yang telah bekerja sama dengan perusahaan, akan mengirimkan tagihan biaya pengiriman barang dalam berkas Excel setiap bulannya. Berkas tersebut akan melalui proses validasi oleh divisi operasional, proses ini meliputi pengecekan terhadap data jarak dan harga dalam berkas tagihan secara manual. Pengecekan dilakukan pada data

jarak dan harga saja, karena sistem hanya menangani transaksi dari *third party logistic* yang tidak menghitung berat dalam pengiriman, seperti: GoSend dan Grab.

Proses pengecekan manual saat ini dilakukan dengan cara mengambil titik *longitude latitude* lokasi asal dan tujuan pengiriman yang terdapat pada berkas. Kemudian dimasukkan ke Google Maps untuk mendapatkan jarak antara lokasi asal dan tujuan. Data jarak yang didapat melalui sistem Google Maps digunakan untuk menghitung total harga. Jarak dan harga yang didapatkan dari proses perhitungan manual ini akan dibandingkan dengan data jarak dan harga yang terdapat dalam berkas tagihan pengiriman. Proses validasi ini perlu dilakukan agar jangan sampai perusahaan harus membayarkan tagihan ongkos kirim yang lebih besar daripada seharusnya.

Data hasil proses validasi yang dilakukan oleh divisi operasional akan diperiksa oleh *supervisor*. Apabila telah valid, data tersebut akan diteruskan ke divisi keuangan untuk dilakukan pengecekan. Jika terdapat data yang tidak valid dalam proses pengecekan oleh *supervisor* ataupun divisi keuangan, maka data akan dikembalikan kepada divisi operasional untuk dilakukan proses validasi kembali. Jumlah data yang diterima oleh divisi operasional dari *third party logistic* adalah lebih dari 10.000 data setiap bulan.

Kendala yang terjadi pada divisi operasional adalah proses validasi yang dilakukan saat ini tidak efisien, karena proses validasi pada suatu data akan dilakukan berkali-kali sebelum dapat dikatakan valid. Kendala lain yang terjadi adalah terdapat data-data yang terlewat karena kesalahan faktor manusia. Seiring berjalannya waktu, data transaksi yang dikelola oleh perusahaan akan semakin besar. Hal ini menjadikan tantangan berat bagi karyawan divisi operasional, untuk mencapai target keseluruhan proses validasi. Oleh karena itu, penelitian ini akan membangun sebuah sistem yang berguna untuk divisi operasional melakukan proses validasi jarak dan harga dengan lebih efektif dan efisien.

Sistem akan mengelola data menggunakan *batch processing*, dengan *framework* yaitu Spring Batch untuk menjalankan fungsi-fungsinya. *Batch processing* didefinisikan sebagai pengelolaan terhadap data tanpa interaksi atau interupsi (Minella, 2011). *Batch processing* dapat membantu sistem dalam mengelola data yang besar dengan membagi data tersebut ke dalam suatu batch atau tahap. *Batch processing* dibutuhkan oleh sistem untuk membaca data dalam jumlah yang besar pada sebuah berkas Excel, melakukan proses terhadap data tersebut, dan

menyimpan ke dalam basis data secara bertahap. Jika data dalam jumlah besar dilakukan proses dan disimpan secara bersamaan, maka akan terjadi kemungkinan *server* tidak kuat sehingga mengakibatkan aplikasi berhenti bekerja. Namun, *batch processing* akan mencegah hal tersebut dengan proses *batching* yang membagi data-data tersebut sebelum diproses dan disimpan ke dalam basis data.

Bahasa pemrograman yang digunakan dalam pembangunan sistem adalah bahasa Java, dengan menerapkan Spring Framework. Spring Framework digunakan karena telah menyediakan ORM (*Object Relational Mapping*), layer abstraksi seperti JDBC, dan memungkinkan mengembangkan aplikasi menggunakan DTO (*Data Transfer Object*) yang memudahkan sistem untuk membaca hasil JSON dari API eksternal seperti Google Maps API. VueJs merupakan *framework javascript* yang dipilih dalam pengembangan *front-end* sistem. VueJs dikenal sebagai *framework* progresif, yang dapat beradaptasi dengan kebutuhan pengembang dan mudah digunakan (Copes, 2018). DBMS yang digunakan pada pengembangan sistem adalah PostgreSQL, dimana dapat menghasilkan UUID (*Universally Unique Identifier*) yang digunakan sebagai ID dalam tabel basis data.

Sistem validasi ini akan dibangun berbasis *website* yang bertujuan untuk menyesuaikan dengan kebutuhan pengguna dan memudahkan karyawan dalam melakukan akses. Hasil yang diharapkan setelah sistem dibangun adalah proses validasi yang dilakukan oleh divisi operasional dapat berjalan secara lebih efisien, mempercepat proses validasi, dan mudah digunakan oleh pengguna.

BAB 2

TINJAUAN PUSTAKA

2.1. Penelitian Terdahulu

Penelitian – penelitian terdahulu tentang penerapan *batch processing* pada sebuah sistem telah berhasil dilakukan. Hal ini membuktikan bahwa *batch processing* dapat diterapkan untuk mempercepat pengolahan data dalam jumlah besar. Namun, penerapan pada sebuah sistem validasi masih jarang ditemukan.

Pada penelitian mengenai implementasi *online testing* dengan *batch processing system* berbasis *website*, dijelaskan bahwa adanya kebutuhan terhadap kecepatan untuk melakukan pemrosesan dan pengelolaan data yang besar, terkhusus pada tes yang bersifat *online*. Tes yang dilakukan secara *online* akan melibatkan banyak peserta, berarti juga akan menghasilkan banyak jawaban yang harus diproses. Bagaimana merancang suatu sistem yang dapat melakukan proses terhadap data peserta yang besar dengan efisien dan memastikan data tersebut valid sebelum dimasukkan ke dalam basis data. Pengujian sistem dilakukan terhadap 50–100 peserta dengan contoh soal yang berbeda. Sistem *batch processing* membantu dalam mempercepat proses implementasi *online testing*, tingkat kesalahan dalam pengacakan soal jauh lebih akurat, dan tepat sasaran (Bagir, 2016).

Pada penelitian mengenai sistem informasi penghasil berkas laporan menggunakan metode *batch processing*, dijelaskan bahwa banyak perusahan yang membutuhkan laporan-laporan untuk melakukan evaluasi terhadap suatu hal. Proses pembuatan laporan di beberapa perusahaan masih dilakukan secara manual, namun pada penelitian ini pembuatan laporan akan dilakukan secara otomatis oleh sistem. Laporan yang dikelola oleh perusahaan berupa sebuah data yang besar, sehingga untuk melakukan eksekusi data laporan tersebut sistem akan dibantu oleh metode *batch processing* untuk mengoptimalkan sistem. Hasil dari penelitian ini adalah aplikasi penghasil laporan dalam berkas yang dapat mempercepat proses pembacaan laporan melalui metode *batch processing* (Syahputra, 2018).

Dalam penelitian-penelitian tersebut masih menggunakan data yang dimasukkan ke sistem secara satu per satu. Melalui penelitian ini, akan

mempermudah pengguna dalam memasukkan data dengan membuat sistem yang mampu menerima masukan data berupa berkas, yaitu mengunggah berkas Excel ke dalam sistem. Sehingga pengguna tidak perlu memasukkan data satu per satu ke dalam sistem.

2.2. Electronic Commerce

Electronic Commerce atau biasa dikenal sebagai *e-commerce*, mengacu pada kegiatan jual beli barang atau jasa melalui internet. *E-commerce* adalah perdagangan produk atau layanan menggunakan jaringan komputer, seperti internet. *E-commerce* berbasis pemrosesan data, termasuk tulisan, suara, dan gambar. Transaksi bisnis yang terjadi pada *e-commerce* adalah *business-to-business* (B2B), *business-to-customer* (B2C), *customer-to-customer* (C2C) atau *customer-to-business* (C2B) (Shahriari, *et al*, 2015). Melalui pertumbuhan ekonomi dan perkembangan teknologi, proses transaksi dan jual beli yang dilakukan secara tradisional, berubah dengan menggunakan *e-commerce* dikarenakan akses yang cepat, pilihan barang atau jasa lebih luas, dan cakupan yang luas. Alasan pemanfaatan *e-commerce* pada bisnis adalah memperluas jangkauan, mengikuti kebutuhan bisnis yang rumit, memahami pelanggan, menganalisis pasar, dan mempercepat balik modal.

2.3. Validasi Data

Validasi data merupakan sebuah komponen penting untuk sistem manajemen data akhir-akhir ini. Proses validasi data akan membantu dalam menghindari duplikasi data dan pemasukan data yang salah ke dalam basis data. Validasi data adalah sebuah proses yang memastikan pengiriman data ke program, aplikasi, dan layanan yang menggunakannya. Validasi data digunakan untuk melakukan pengecekan pada integritas dan validasi data yang dimasukkan ke perangkat lunak. Validasi data digunakan untuk memastikan data sesuai dengan persyaratan tertentu dan tolak ukur kualitas (Thiruthanigesan & Thiruchchelvan, 2016).

2.4. Batch Processing

Beberapa aplikasi pada perusahaan besar pada saat ini, mengerjakan perintah-perintah yang dapat dieksekusi tanpa tampilan atau *user interface* untuk menjalankannya. Perintah-perintah ini biasa dijalankan secara periodik dan

melakukan proses pada data yang berjumlah besar. Contoh yang dapat ditemukan adalah proses tagihan, pembuatan laporan, konversi format data, dan *image processing*. *Batch processing* didefinisikan sebagai proses yang dilakukan terhadap sejumlah kasus secara bersamaan (Martin, *et al*, 2015).

Batch Processing merupakan sebuah solusi bagi proses yang melibatkan data yang besar. *Batch Processing* merupakan sebuah proses dari pemrosesan data tanpa interaksi dan interupsi (Minella, 2011). Cara kerja dari *batch processing* adalah mengelompokkan sebuah data ke dalam himpunan tertentu, yang selanjutnya akan diatur jumlah data yang akan disimpan ke dalam basis data. Pada proses penulisan data ke dalam basis data, melalui *batch processing*, data yang sudah dikelompokkan tersebut akan diperiksa kembali sebelum disimpan ke dalam basis data. Jika terdapat data yang tidak valid, maka data tersebut akan masuk ke dalam *error report*. Sehingga hasil data yang disimpan ke dalam basis data adalah data-data yang sudah valid dan tidak terdapat kesalahan.

Terdapat sedikitnya enam alasan menggunakan *Batch Processing* yaitu *maintability*, fleksibilitas, *scalability*, *development resouces*, *support*, dan biaya. Setiap kali dijalankan *batch processing* akan mengambil atau membaca sekumpulan data yang akan langsung melakukan proses sehingga memori yang dibutuhkan tidak terlalu besar. Penggunaan memori yang baik, akan memberikan dampak kepada kinerja aplikasi, sehingga proses waktu eksekusi yang diberikan bisa lebih optimal.

2.5. Spring Framework

Pengembangan aplikasi perangkat lunak tanpa disertai *tool* dan teknologi yang baik, akan mengurangi produktifitas dari pembuatan aplikasi. Spring Framework merupakan sebuah *framework open source* berbasis Java yang diciptakan untuk memudahkan pengembangan perangkat lunak Java. Sebagai *framework* yang berbasis Java, maka Spring juga menerapkan kerangka berbasis objek. Spring Framework memudahkan pengembang dalam melakukan pengembangan baik untuk aplikasi *website*, aplikasi-aplikasi yang berdiri sendiri, aplikasi *enterprise*, dan lain sebagainya. Spring Framework menyederhanakan pengembang *enterprise* Java. (Walls, 2014).

Selain ringan, Spring Framework memiliki banyak *framework* pendukung. Ini merupakan keunggulan dimana Spring Framework tidak bertujuan untuk

menggantikan fungsi *framework* tertentu, namun menjadi dasar bagi *framework* lainnya. Spring framework juga mendukung beberapa modul seperti Spring Security, Spring Boot, dan Spring Batch (Martin, *et al*, 2015).

2.6. Spring Boot

Spring Boot merupakan sebuah projek yang dibangun di atas Spring Framework. Melalui Spring Boot dapat dilakukan konfigurasi secara otomatis yang memungkinkan bagi Spring mendeteksi jenis aplikasi yang sedang dibangun. Spring Boot menyediakan fungsi-fungsi yang dibutuhkan dalam pengembangan aplikasi Java sehingga pengembang dipermudah dalam melakukan pengembangan sistem. Spring Boot merupakan *framework* berbasis Java yang digunakan untuk membuat sebuah *micro service* (Walls, 2014).

Spring Boot didesain untuk menghindari konfigurasi XML yang kompleks dan mengurangi waktu untuk mengembangkan aplikasi. Spring Boot menyediakan cara yang fleksibel bagi konfigurasi seperti Java Beans, konfigurasi-konfigurasi XML, dan transaksi yang terjadi dalam basis data. Dalam Spring Boot semuanya sudah dikonfigurasi secara otomatis, sehingga tidak perlu menghabiskan waktu untuk membuat konfigurasi satu per satu.

Spring Boot digunakan terutama dalam pembuatan RESTful API dan halaman *website*. Salah satu kelebihan dari penggunaan Spring Boot adalah penggunaan JSON API dalam berkomunikasi antara *server*. Hal ini menguntungkan dalam pembangunan aplikasi berbasis *website*, dimana saat ini JSON banyak digunakan sebagai format pertukaran data antar sistem (Nguyen, 2018).

2.7. Spring Batch

Banyak aplikasi dalam *enterprise* menjalakan *batch processing* dalam melakukan bisnis operasi. Bisnis operasi ini meliputi otomasi, proses yang kompleks dengan data yang besar tanpa interaksi. Fitur yang diterapkan oleh Spring Batch termasuk validasi data, format keluaran, kemampuan untuk menerapkan aturan bisnis yang kompleks dengan cara yang dapat digunakan kembali, dan kemampuan untuk menangani kumpulan data besar (Minella, 2011). Spring Batch menyediakan fungsi-fungsi yang dibutuhkan pengembang dalam mengembangkan aplikasi *enterprise* menggunakan *batch processing*. Spring

Batch merupakan sebuah *framework* yang besifat *open source* untuk melakukan *batch processing* dengan melakukan eksekusi terhadap berbagai *job*.

Spring Batch menyediakan kelas dan API untuk membaca atau menuliskan *resource-resource*, mengelola transaksi, melakukan proses statistik, mengulang *job*, dan teknik partisi untuk memproses data yang besar. Spring Batch melakukan proses pada *item-item* ke dalam *chunk*. Cara kerja Spring Batch adalah sebuah *job* akan melakukan proses baca (*read*) dan menuliskan (*write*) *item-item* ke dalam sebuah *chunk* yang lebih kecil. *Chunk* atau biasa dikenal sebagai *chunk processing* merupakan algoritma berorientasi *batch* spesifik yang berisi himpunan pada sebuah alur eksekusi (Cogoluegnes, *et al*, 2011).

BAB 3

MASALAH, TUJUAN, DAN MANFAAT PENELITIAN

3.1. Perumusan Masalah

Dalam penelitian ini dapat dijabarkan perumusan masalah yang ada, yaitu adalah bagaimana cara membangun sistem validasi jarak dan harga menggunakan metode *batch processing* pada divisi operasional untuk memberikan perbandingan terhadap perbedaan jarak dan harga.

3.2. Tujuan Penelitian

Tujuan dari diadakannya penelitian, perancangan, dan pembangunan sistem validasi jarak dan harga pada divisi operasional adalah untuk membangun sistem validasi yang akan memberikan perbandingan terhadap perbedaan jarak dan harga.

3.3. Manfaat Penelitian

Manfaat penelitian ini adalah sebagai berikut :

1. Hasil utama dari penelitian ini adalah menghasilkan sebuah *protoype* perangkat lunak yang merupakan sebuah sistem yang siap untuk diimplementasikan.
2. Hasil yang lain adalah sebuah *paper* penelitian yang dipublikasikan minimal di jurnal internasional.

BAB 4

METODE PENELITIAN

Metode penelitian pada bagian ini membahas langkah-langkah yang dilakukan dari awal hingga penelitian selesai dan mendapatkan hasil yang diinginkan. Langkah – langkah tersebut terdiri dari :

1. Observasi

Observasi dilakukan untuk mendapatkan data dan informasi mengenai proses bisnis yang terjadi dalam divisi operasional dengan cara melakukan pengamatan secara langsung ke perusahaan PT. XYZ.

2. Wawancara

Kegiatan wawancara dilakukan dengan mengajukan pertanyaan langsung kepada divisi operasional perusahaan PT. XYZ untuk mengetahui permasalahan, kebutuhan pengguna, dan menyesuaikan data dengan kondisi sesungguhnya. Wawancara juga bertujuan untuk mengetahui kebutuhan sistem, yang akan diwujudkan dalam perancangan sistem.

3. Analisis

Pada tahap ini peneliti akan menggali tiap permasalahan yang menjadi perhatian oleh divisi operasional, melalui rangkaian proses pengumpulan data. Peneliti akan mendapatkan gambaran mengenai kebutuhan terhadap program yang akan dibuat, baik dari sisi fungsionalitas dan non-fungsionalitas. Proses analisis akan dilakukan secara mendalam untuk mendapatkan spesifikasi kebutuhan perangkat lunak yang sesuai, untuk membantu menyelesaikan permasalahan pengguna melalui hasil program yang dibuat. Pada tahap ini, seluruh spesifikasi kebutuhan perangkat lunak akan didokumentasikan. Dokumentasi akan digunakan sebagai pegangan bagi penulis dalam membangun sistem dan juga gambaran bagi pengguna mengenai aplikasi seperti apa yang akan dibangun. Hasil dari analisis akan didokumentasikan menjadi dokumen Spesifikasi Perangkat Lunak (SKPL).

3. Perancangan

Pada tahap perancangan, peneliti membangun rancangan komponen atau kerangka perangkat lunak, berdasarkan kebutuhan serta data yang sudah dianalisis melalui proses analisis. Proses perancangan akan terdiri dari: tampilan antar muka, struktur data, basis data, arsitektur aplikasi, *use case*, dan diagram relasi entitas. Tahap ini akan menafsirkan hasil dari kebutuhan perangkat lunak ke gambaran desain, sehingga dapat diterapkan pada aplikasi. Hasil dari perancangan akan didokumentasikan menjadi Deskripsi Pengembangan Perangkat Lunak (DPPL).

5. Pengkodean

Pada tahap pengkodean akan dilakukan proses menterjemahkan hasil perancangan aplikasi yang telah dibuat, ke dalam kode-kode bahasa pemrograman yang dapat dimengerti oleh komputer. Melalui tahap ini akan menghasilkan sistem untuk menangani kebutuhan-kebutuhan pada divisi operasional, sehingga dapat menggantikan proses validasi yang dilakukan secara manual. Metode yang digunakan adalah *Batch Processing* yang diimplementasikan melalui Spring Batch dengan tahapan sebagai berikut:

a. Step

Step merupakan sebuah objek domain yang terdiri dari tahapan sekuensial pada suatu *batch* dan berisi semua informasi yang diperlukan, untuk menentukan dan mengontrol *batch processing*. Tahapan sekuensial tersebut terdiri dari tiga fungsi yaitu *ItemReader*, *ItemProcessor*, dan *ItemWriter*. *ItemReader* adalah pembacaan data dan menyiapkan data tersebut dari berbagai tipe masukan yang berbeda. Setiap data yang telah dibaca, akan diproses, proses ini dilakukan pada fungsi *ItemProcessor*. *ItemWriter* adalah penulisan hasil keluaran dan menyimpan dalam basis data.

b. Job

Job adalah sebuah abstraksi eksplisit yang berisi konfigurasi atau pengaturan dalam menjalankan proses *batch processing* melalui Spring Batch. *Job* akan berisi pemanggilan *Step* yang sudah dibuat dan digunakan untuk menjalankan atau mematikan proses atau *Step*.

6. Pengujian

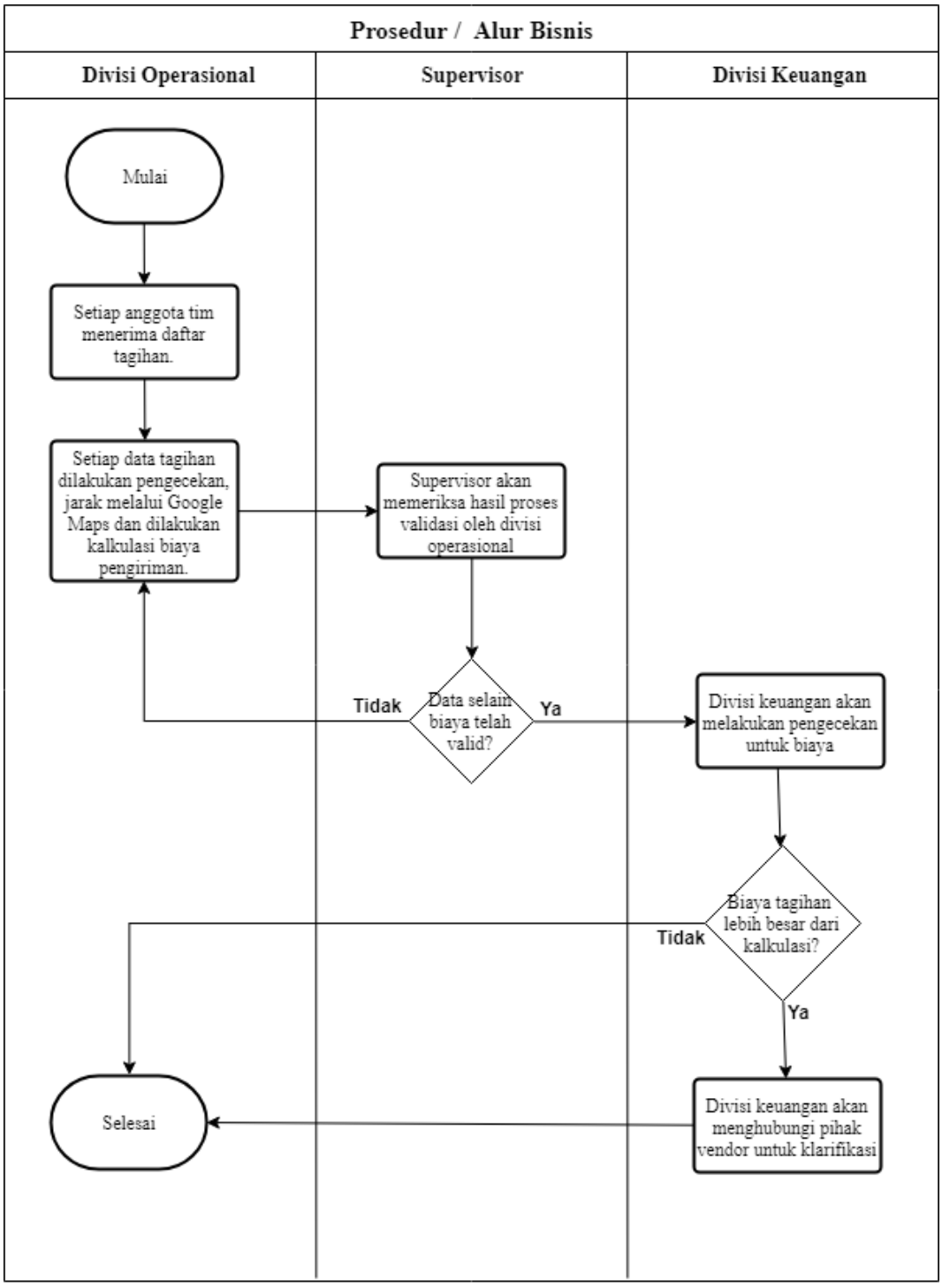
Tahap ini akan dilakukan pengujian perangkat lunak, yang telah dibuat melalui rangkaian proses tahap-tahap sebelumnya. Pengujian dilakukan untuk menguji fungsionalitas dari perangkat lunak, untuk mengetahui jika terdapat kesalahan atau *bug* dalam perangkat lunak yang telah dibuat. Selain memastikan perangkat lunak bebas dari kesalahan, proses pengujian dilakukan agar memastikan hasilnya sesuai dengan kebutuhan yang sudah dispesifikasi sebelumnya. Proses pengujian akan dilakukan dengan menggunakan metode *black box*, yaitu dengan menguji fungsionalitas yang ada dalam aplikasi dan mengamati hasil eksekusi perangkat lunak.

BAB 5

HASIL DAN PEMBAHASAN

5.1. Analisis Sistem

5.1.1. Proses Validasi Manual



Gambar 5.1. Diagram Alur Proses Validasi Manual

Pada Gambar 5.1. terdapat tiga aktor yang berperan dalam proses validasi secara manual yaitu divisi operasional, supervisor, dan divisi keuangan. Divisi operasional merupakan tim yang bertanggung jawab dalam segala proses operasional dalam perusahaan, salah satu proses tersebut adalah proses validasi. Proses validasi merupakan proses pengecekan jarak dan harga pada suatu berkas yang terdiri dari ribuan data tagihan oleh vendor. Divisi operasional saat ini terdiri dari tujuh anggota, setiap anggota memiliki target data untuk diproses kurang lebih 500 data per hari. Proses validasi dimulai ketika setiap anggota telah menerima data yang akan dilakukan pemeriksaan berupa berkas (.xlsx). Setiap data memiliki titik *longitude* dan *latitude* yang berfungsi untuk mencari jarak melalui Google Maps.

Jarak yang telah didapat akan digunakan untuk melakukan kalkulasi biaya pengiriman berdasarkan aturan yang telah disepakati oleh perusahaan dan vendor. Data jarak melalui Google Maps dan biaya yang telah dihitung akan dilakukan perbandingan pada data yang ditagihkan oleh vendor. Selanjutnya, supervisor akan melakukan pemeriksaan terhadap data-data yang telah diproses oleh divisi operasional. Jika terdapat kesalahan selain biaya, seperti kesalahan perhitungan jarak, maka data akan dikembalikan kepada divisi operasional untuk dilakukan pemeriksaan. Jika kesalahan terdapat pada bagian biaya, yaitu biaya yang ditagihkan oleh vendor lebih besar dari perhitungan manual maka divisi keuangan akan menghubungi vendor untuk klarifikasi. Proses ini akan dilakukan secara berulang hingga mendapatkan data yang valid.

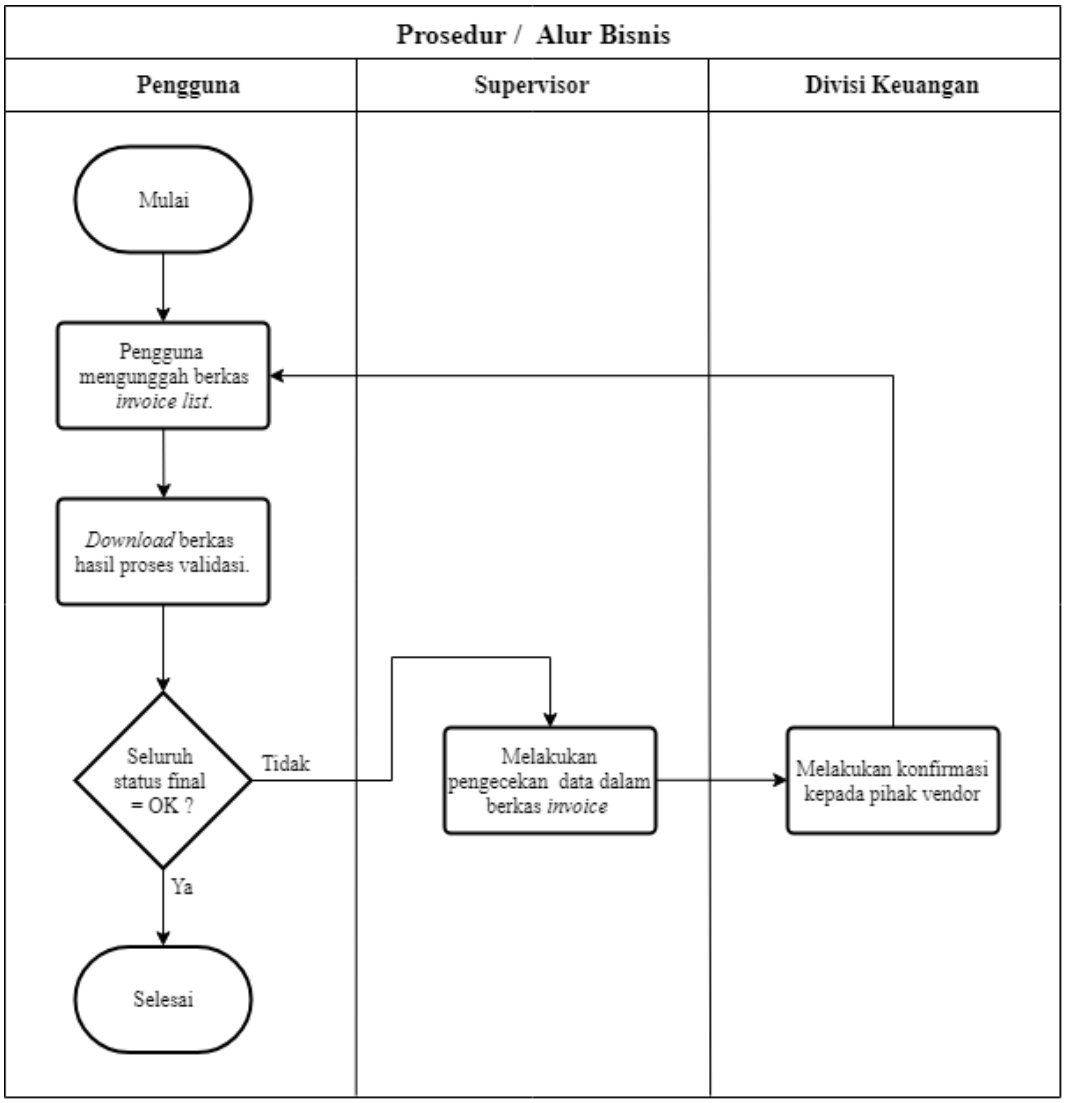
5.1.2. Lingkup Masalah

Sistem validasi jarak dan harga (SIJAHA) adalah perangkat lunak untuk manejemen *third party logistic* dan melakukan validasi terhadap tagihan dari *third party logistic*. SIJAHA dibuat untuk membuat proses validasi menjadi lebih efisien. Pengguna hanya perlu mengunggah berkas (.xlsx) yang berisi tagihan *invoice* dari *third party logistic* dan aplikasi akan melakukan proses validasi yang selanjutnya menghasilkan laporan dalam berkas (.xlsx). Program ini dirancang dalam bahasa pemrograman Java dan membutuhkan koneksi internet untuk menjalankannya.

Manfaat yang diperoleh oleh divisi operasional pada perusahaan dengan diterapkan sistem ini adalah proses validasi yang selama ini dilakukan secara

manual, dimudahkan dengan menggunakan sistem dalam proses perhitungan validasi. Dengan jumlah transaksi yang terus meningkat, proses validasi akan lebih efisien dan efektif jika menggunakan sistem.

5.1.3. Alur Sistem



Gambar 5.2. Diagram Alur Bisnis Produk

Pada Gambar 5.2. terdapat tiga aktor yang berperan dalam proses bisnis yaitu pengguna, supervisor, dan divisi keuangan. Pengguna merupakan pegawai dari divisi operasional yang menangani proses validasi suatu berkas yang berisi kumpulan *invoice* pengiriman barang. Supervisor merupakan orang yang menjadi penanggung jawab untuk divisi operasional dan divisi keuangan merupakan bagian yang menangani keuangan pada perusahaan. Pada diagram alur bisnis pengguna akan mengunggah berkas (.xlsx) yang berisi kumpulan *invoice* untuk dilakukan proses validasi. Pengguna akan mengunduh berkas pelaporan hasil

proses validasi dan melakukan pemeriksaan terhadap setiap status final pada suatu *invoice*. Jika terdapat status final selain OK dengan data yang tidak valid berkaitan dengan aturan operasional, seperti jarak atau perhitungan *rule third party logistic*, maka pengguna akan meneruskan kepada supervisor untuk dilakukan pengecekan. Supervisor akan meneruskan kepada divisi keuangan untuk dilakukan konfirmasi kepada vendor, terkait biaya pengiriman yang berbeda antara perhitungan oleh sistem dan yang terdapat pada tagihan. Data yang telah dilakukan pengecekan akan diunggah kembali hingga seluruh status final adalah OK.

5.1.4. Perspektif Produk

Sistem ini menggunakan DBMS yaitu PostgreSQL, dengan dirancang berbasis objek, dalam bahasa pemrograman Java. Sistem ini membutuhkan konektivitas internet yang digunakan untuk mengakses Google Maps API. Penerapan sistem ini akan membantu menyelesaikan masalah dalam divisi operasional yaitu proses yang selama ini dilakukan secara manual akan dipermudah dengan adanya sistem. Sistem ini akan terdiri dari empat bagian: manajemen aturan *third party logistics*, manajemen *invoice list*, pengolahan data untuk divalidasi, unggah daftar berat barang, dan penyimpanan riwayat dari AWB yang pernah diolah.

Perangkat lunak web SIJAHA berjalan pada *web browser* Chrome dan memanfaatkan *framework* yaitu Spring Boot untuk *back-end* pada web. Proses pengolahan data akan menggunakan Spring Batch yang menyediakan fungsi-fungsi untuk menjalankan *batch processing*. Tampilan pada web menggunakan Vuejs yang memungkinkan antarmuka menjadi *Single Page Aplication* (SPA). Sedangkan untuk lingkungan pemrogramannya menggunakan Intellij IDEA 2017.3.4.

5.1.5. Antarmuka Perangkat Lunak

Perangkat lunak yang dibutuhkan untuk mengoperasikan perangkat lunak SIJAHA di sisi server adalah sebagai berikut :

1. Nama : Tomcat 7 atau diatasnya
   Sumber : Apache
   Deskripsi : Sebagai *web server* untuk menjalankan SIJAHA.
2. Nama : PostgreSQL 9.6

Sumber : PostgreSQL Global Development Group

Deskripsi : Sebagai *data base management system* yang digunakan untuk penyimpanan data pada basis data.

3. Nama : Apache POI

   Sumber : Apache

   Deskripsi : Sebagai API pada bahasa pemrograman Java yang memungkinkan untuk membuat dan memodifikasi berkas MS. Office

4. Nama : Rest Template

   Sumber : Spring

   Deskripsi : Sebagai *client-side* akses HTTP untuk API eksternal.

Sedangkan perangkat lunak yang dibutuhkan untuk mengoperasikan perangkat lunak SIJAHA di sisi klien adalah sebagai berikut :

1. Nama : Google Chrome

   Sumber : Google, Inc

   Deskripsi : Sebagai *tools* antarmuka tampilan yang digunakan SIJAHA.

5.1.6. Kebutuhan Fungsional

Perangkat lunak web SIJAHA mempunyai fungsionalitas sistem sebagai berikut:

1. Fungsi Login

   Fungsi *login* merupakan fungsi yang memungkinkan pengguna untuk memasuki sistem dan menggunakan fungsionalitas yang tersedia dalam sistem.

2. Fungsi Kelola Aturan *Third Party Logistics*

   Fungsi kelola aturan *third party logistics* adalah fungsi yang digunakan oleh pengguna melakukan pengelolaan data aturan *third party logistics*. Fungsi ini terdiri dari:

   a. Fungsi Tambah Aturan

      Fungsi ini digunakan oleh pengguna untuk menambahkan data aturan baru. Data aturan yang ditambahkan berupa *logistic code*, formula jarak, *rule type*, harga, dan jarak.

   b. Fungsi Ubah Aturan

   c. Fungsi ini digunakan oleh pengguna untuk mengubah data aturan berupa *logistic code*, formula jarak, *rule type*, harga, dan jarak.

   d. Fungsi *Soft Delete* Aturan

Fungsi ini digunakan oleh pengguna untuk melakukan *soft delete* atau menon-aktifkan status dari suatu aturan.

e. Fungsi Cari Aturan

Fungsi ini digunakan oleh pengguna untuk mencari data aturan berdasarkan inputan pengguna.

f. Fungsi Melihat Daftar *Third Party Logistics*

Fungsi ini memungkinkan pengguna untuk melihat daftar *third party logistics* yang tersedia.

g. Fungsi Mencari *Third Party Logistics*

Fungsi ini memungkinkan pengguna untuk melakukan pencarian data *third party logistics* berdasarkan masukan pengguna.

3. Fungsi Kelola *Invoice*

Fungsi kelola *invoice* adalah fungsi yang digunakan oleh pengguna melakukan pengelolaan data *invoice*. Fungsi ini terdiri dari:

a. Fungsi Tambah *Invoice*

Fungsi ini digunakan oleh pengguna untuk menambahkan data *invoice* baru. Data aturan yang ditambahkan berupa *logistic code*, bulan, dan tahun.

b. Fungsi *Soft Delete* Invoice

Fungsi ini digunakan oleh pengguna untuk melakukan *soft delete* atau menon-aktifkan status dari suatu *invoice*.

c. Fungsi Cari *Invoice*

Fungsi ini digunakan oleh pengguna untuk mencari data *invoice* berdasarkan inputan pengguna.

4. Fungsi Validasi Jarak dan Harga

Fungsi ini digunakan oleh pengguna untuk melakukan proses validasi data suatu *invoice*. Fungsi ini terdiri dari:

a. Fungsi Unggah Berkas *Invoice*

Fungsi ini digunakan oleh pengguna untuk melakukan unggah berkas (.xlsx) invoice yang berisi ratusan hingga ribuan data yang akan dilakukan proses validasi oleh sistem.

b. Fungsi Melakukan *Approve*

Fungsi ini digunakan oleh pengguna untuk melakukan aksi *approve* jika data sudah valid dan disetujui oleh supervisor.

c. Fungsi Melakukan *Reject*

Fungsi ini digunakan oleh pengguna untuk melakukan aksi *reject* jika data dianggap tidak disetujui oleh supervisor.

d. Fungsi Menampilkan *Action History*

Fungsi ini memungkinkan pengguna untuk melihat informasi riwayat *action* yang diberikan pada suatu berkas *invoice list*.

e. Fungsi Menampilkan *Upload History*

Fungsi ini memungkinkan pengguna untuk melihat informasi riwayat *upload* berkas yang dilakukan pada suatu berkas *invoice list*.

5. Fungsi Kelola Berat Barang

Fungsi kelola berat barang adalah fungsi yang digunakan oleh pengguna melakukan pengelolaan data berat barang. Fungsi ini terdiri dari:

a. Fungsi Unggah Berkas

Fungsi ini digunakan oleh pengguna untuk mengunggah berkas (.xlsx) yang berisi data daftar berat barang ke dalam sistem.

b. Fungsi Hapus Berat Barang

Fungsi ini digunakan oleh pengguna untuk menghapus data suatu berat barang.

c. Fungsi Cari Berat Barang

Fungsi ini digunakan oleh pengguna untuk mencari data berat barang berdasarkan masukan pengguna.

6. Melihat Informasi Riwayat AWB

Fungsi ini memungkinkan pengguna untuk melihat informasi riwayat suatu AWB.

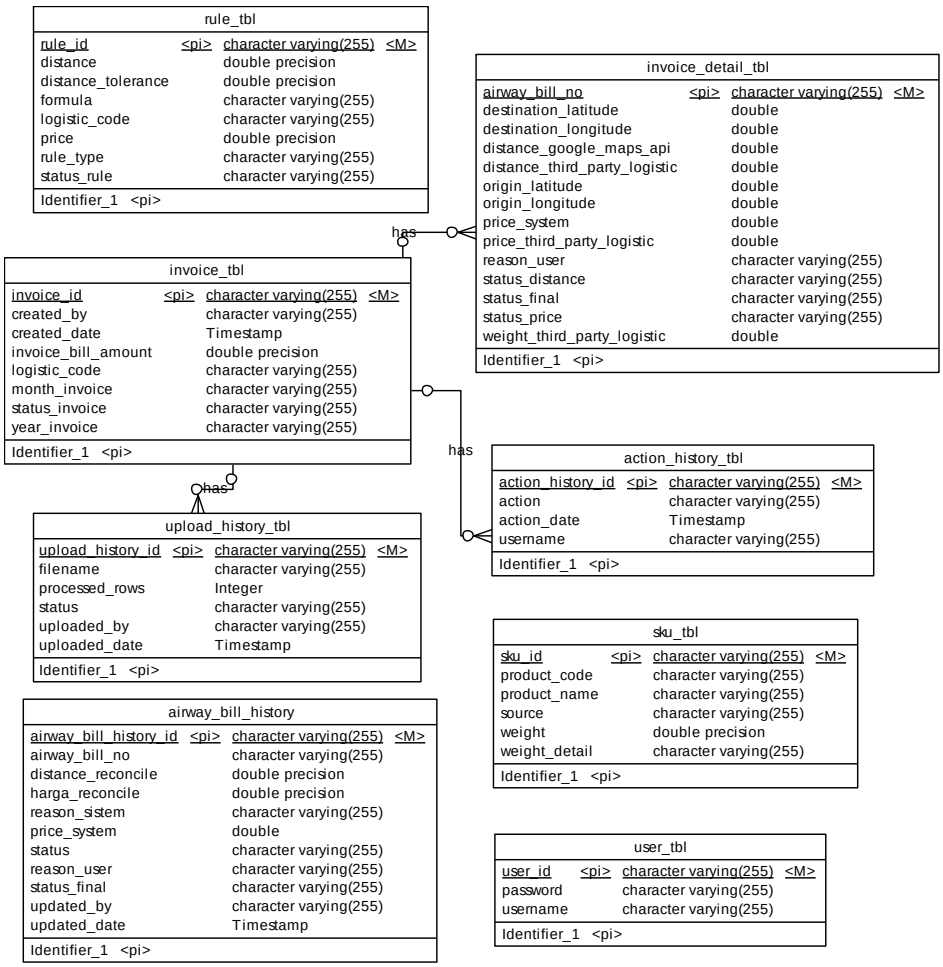
7. Fungsi Membuat *Distance Tolerance*

Fungsi ini memungkinkan pengguna untuk menambahkan data d*istance tolerance* berdasarkan suatu *logistic code*. Data yang ditambahkan berupa *logistic code* dan jumlah persen toleransi yang diberikan.

Gambar 5.3. Use Case Diagram SIJAHA

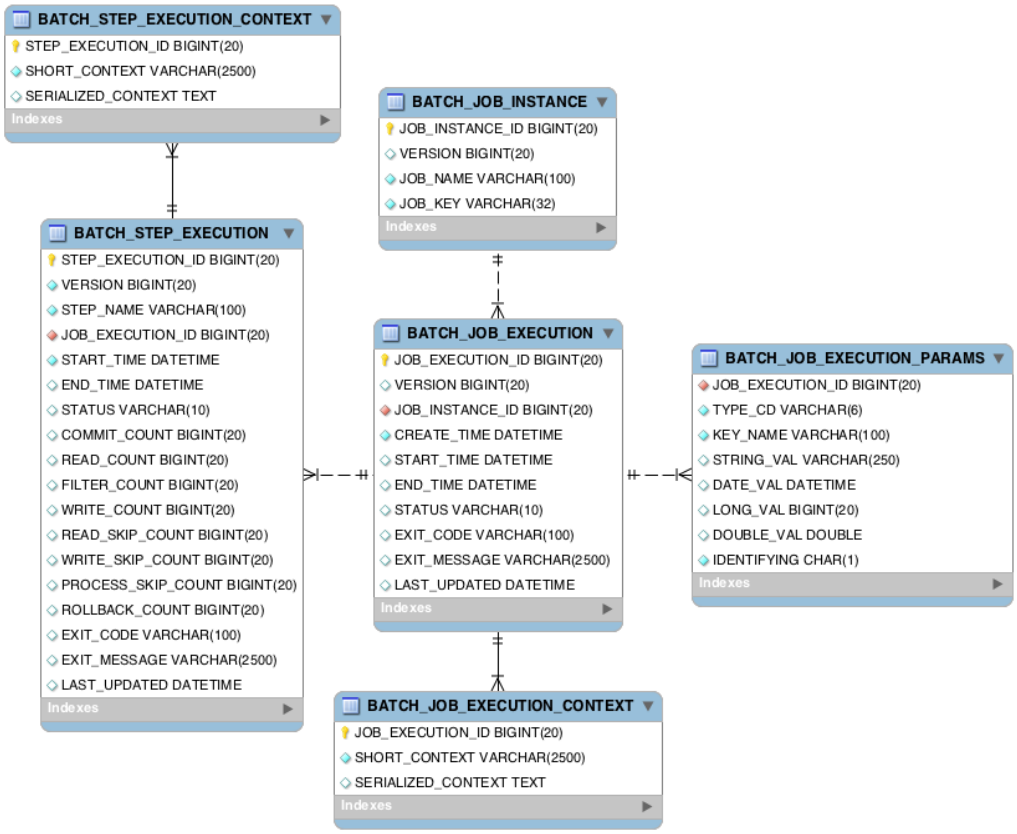
Gambar 5.3. menunjukkan *use case diagram* perangkat lunak SIJAHA. Pada SIJAHA hanya terdapat satu aktor yaitu pengguna yang dapat melakukan *login*, mengelola aturan *third party logistic*, mengelola *invoice*, melakukan validasi jarak dan harga, mengelola berat barang, mengelola riwayat AWB, dan membuat *distance tolerance*.

5.1.7. Entity Relationship Diagram



Gambar 5.4. Entity Relationship Diagram (ERD) Perangkat Lunak SIJAHA

Gambar 5.4. menunjukkan ERD perangkat lunak SIJAHA yang akan menyimpan data-data, yang dibutuhkan oleh sistem dalam melakukan proses bisnis. Analisis sistem secara detail dapat dilihat pada Spesifikasi Kebutuhan Perangkat Lunak (SKPL) yang telah dilampirkan bersama laporan ini.

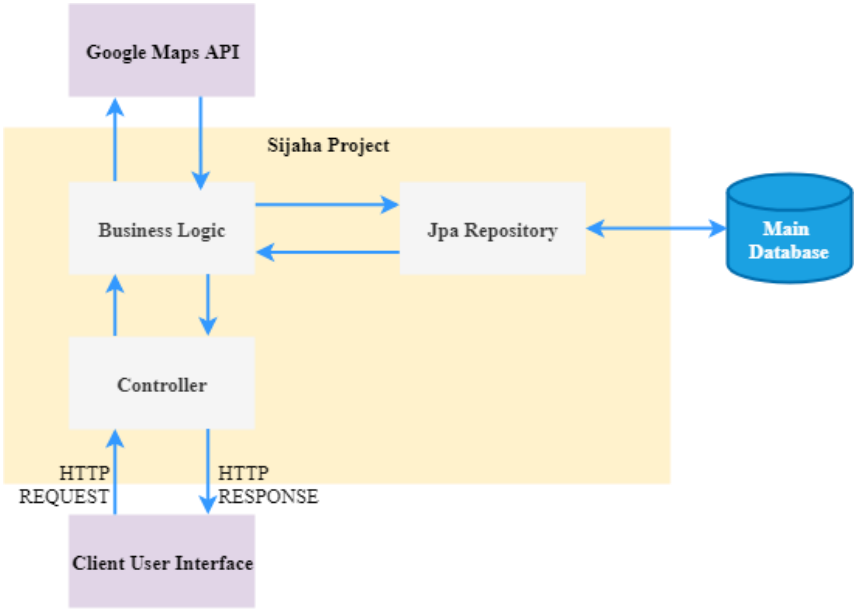


Gambar 5.5. Entity Relationship Diagram (ERD) Spring Batch Perangkat Lunak SIJAHA

Gambar 5.5. menunjukkan ERD Spring Batch perangkat lunak SIJAHA. ERD yang terdapat pada Gambar 5.4. dan Gambar 5.5 terdapat pada basis data yang sama, namun tidak memiliki relasi diantaranya. Hal tersebut disebabkan oleh ERD pada Gambar 5.5., terbuat secara otomatis ketika sistem mengimplementasikan *framework* Spring Batch. Spring Batch membutuhkan tabel pada ERD Gambar 5.5. untuk menyimpan data seperti *Job* dan *Step* ketika *batch processing* dijalankan. ERD pada Gambar 5.4. menangani data-data yang dibutuhkan sistem untuk melakukan proses bisnis.

5.2. Perancangan Sistem

5.2.1. Arsitektur Sistem

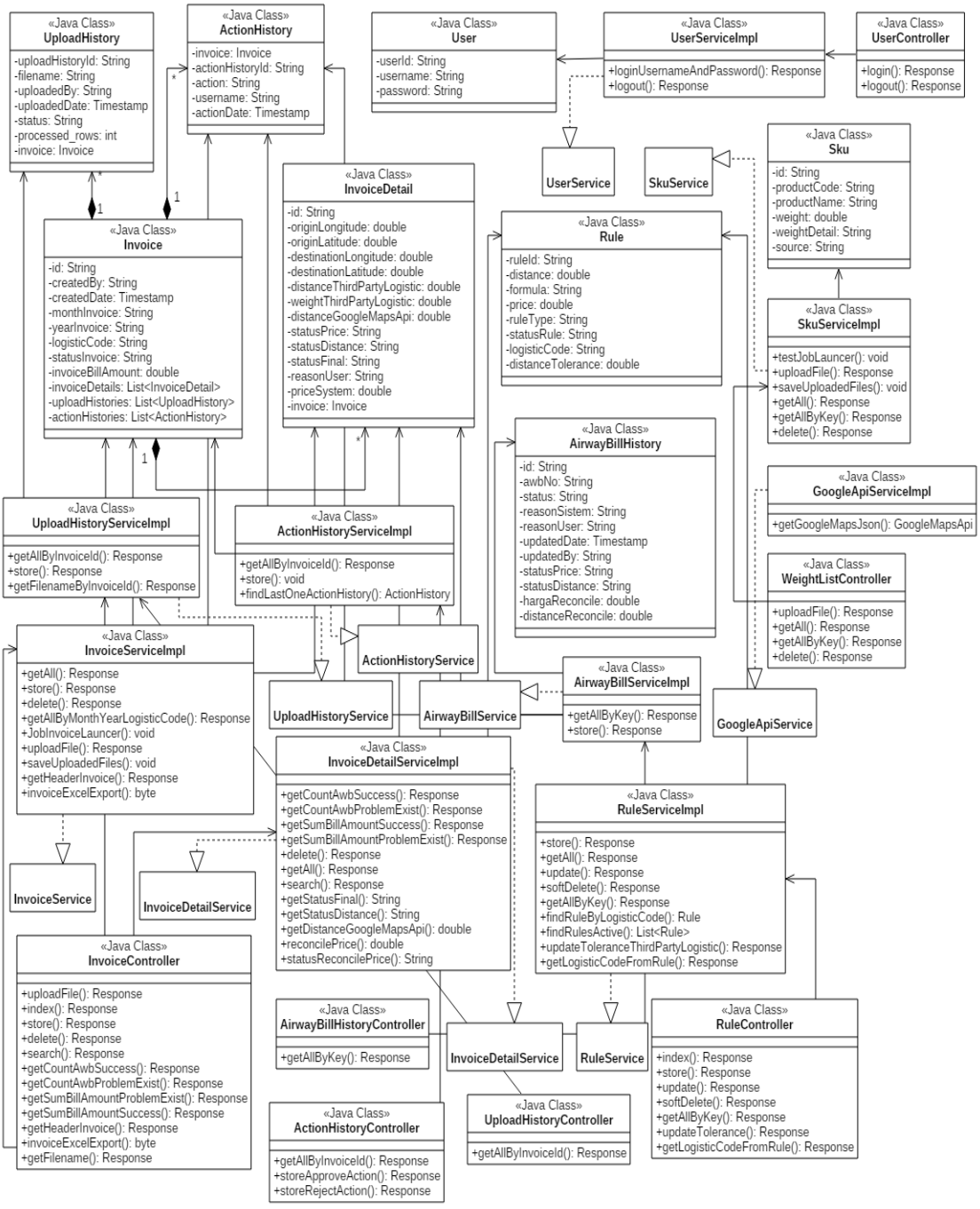


Gambar 5.6. Rancangan Arsitektur Sistem

Gambar 5.6. merupakan arsitektur sistem MVC(*Model View Controller*) yang digunakan oleh SIJAHA. Pertukaran data antara *front-end* dan *back-end* pada sistem menggunakan *Hypertext Transfer Protocol*(HTTP) *request* dan *Representation State* (REST). REST digunakan untuk melakukan *produces* dan *consumes* data dalam bentuk *JavaScript Object Notation*(JSON), untuk melakukan komunikasi pada setiap fungsi akan menggunakan HTTP *request* yang berbentuk *url path*.

Logika bisnis pada *controller* akan memanggil dari *service layer* yang ada, setiap logika bisnis dilakukan pada *service* menggunakan data yang diambil dari *repository*. *Repository* merupakan lapisan yang akan berkomunikasi dengan basis data utama pada sistem. Pada lapisan *service* akan dilakukan logika bisnis dan *consume* data ke Google Maps API berdasarkan suatu titik *longitude* dan *latitude*.

5.2.2. Kelas Diagram



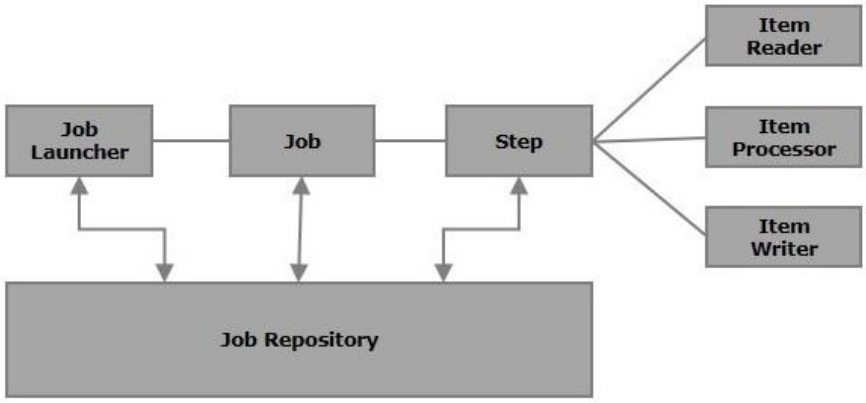
Gambar 5.7. Kelas Diagram

5.3. Implementasi Sistem

SIJAHA merupakan sebuah perangkat lunak yang memiliki fungsi utama yaitu membantu pengguna dalam mengelola barisan data pada suatu berkas (.xlsx) dengan memanfaatkan Google Maps API dan menghasilkan laporan dalam bentuk berkas (.xlsx). Fungsi lain dibutuhkan seperti CRUD untuk *rule* pada suatu

logistic code, *invoice header*, mengunduh berkas daftar berat barang, dan menampilkan riwayat suatu *airwaybill history*. Untuk mendapatkan daftar *logistic code*, sistem akan mengambil data dari API lokal yang disediakan oleh perusahaan.

SIJAHA merupakan perangkat lunak berbasis web yang berjalan dengan *browser* Google Chrome. Aplikasi ini membutuhkan konektivitas internet yang digunakan untuk mengakses Google Maps API. DMBS pada sistem akan menggunakan PostgreSQL dan dirancang berbasis objek, dalam bahasa pemrograman Java. Pengolahan data untuk proses validasi dilakukan dengan cara mengambil data *origin* dan *destination longitude latitude* dari berkas *invoice* yang diunggah. Setelah itu menghitung jarak menggunakan Google Maps API, jika pada *logistic code* terdapat toleransi, maka jarak akan ditambah dengan toleransi yang telah ditentukan. Sistem melakukan pengecekan apakah jarak yang diberikan 3PL berada dibawah atau sama dengan jarak yang telah dikalkulasi oleh sistem. Selanjutnya, sistem akan menghitung biaya dengan rumus *rule* sesuai dengan *logistic code*.

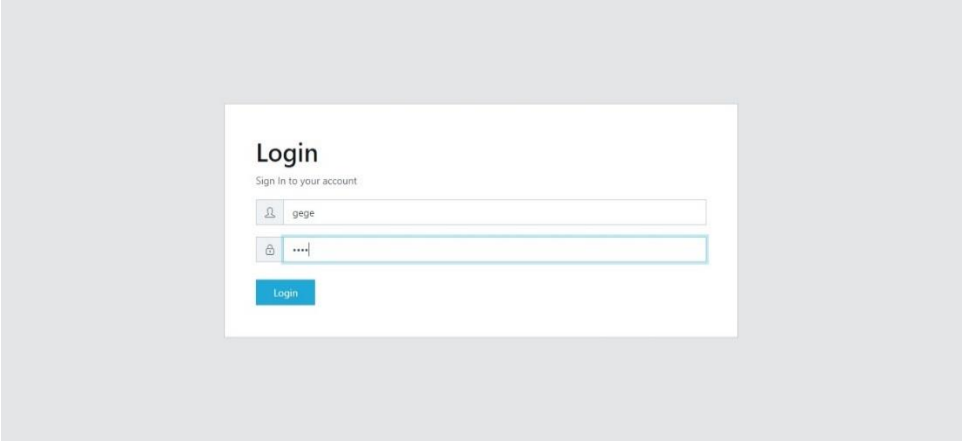


Gambar 5.8. Komponen-Komponen Spring Batch

SIJAHA menggunakan *framework* yaitu Spring Batch untuk mengelola fungsi-fungsi dari *batch processing*. Gambar 5.8. merupakan komponen dasar penyusun dari Spring Batch yaitu *Job*, *Step* (*Reader, Processor, Writer*), dan *Job Launcher*. *Job* merupakan proses *batch* yang akan dieksekusi melalui *Job Launcher*. *Job* akan memiliki *step* yang berisi tahap dasar dalam proses pengelolaan data seperti membaca, memproses, dan menuliskan data ke dalam basis data. Semua komponen ini digunakan dalam SIJAHA untuk mengelola data dalam berkas (.xlsx). Pertukaran data antara *front-end* dan *back-end* pada sistem

menggunakan *Hypertext Transfer Protocol*(HTTP) *request* dan *Representation State* (REST) digunakan untuk melakukan *produces* dan *consumes* data.

5.3.1. Login



Gambar 5. 9. Antarmuka Login

```
@RestController
@RequestMapping("/api")
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping(
            value = "/login",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON_VALUE,
            consumes = MediaType.APPLICATION_JSON_VALUE
    )
    public Response login(@Valid @RequestBody LoginRequest request){
        return userService.loginUsernameAndPassword(request.getUsername(), request.getPassword());
    }
```

Gambar 5.10. Controller Login

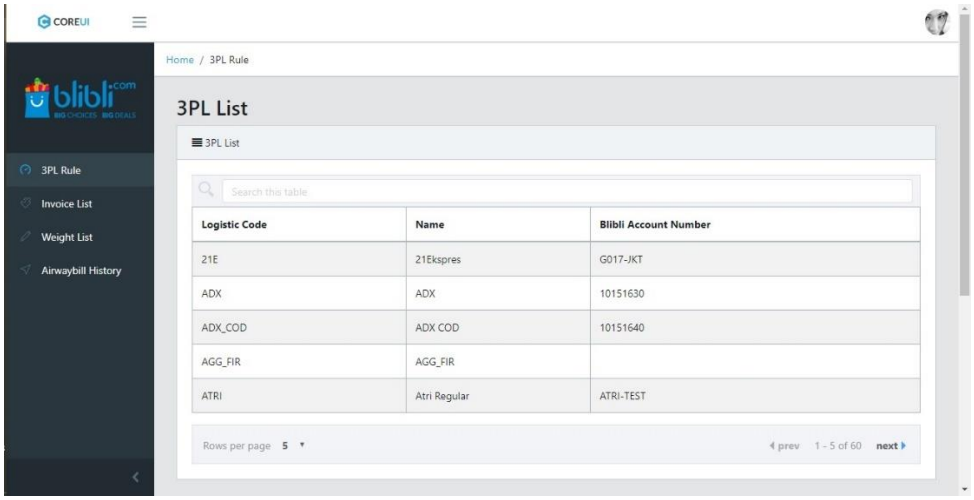
```
    public Response loginUsernameAndPassword(String username, String password){
        User user = userRepository.loginUserWithUsernameAndPassword(username, password);
        if(user != null){
            GlobalVariable.username = user.getUsername();
            return Response.builder()
                    .message("OK")
                    .data(user)
                    .build();
        }
        return Response.builder()
                .message("Failed!")
                .build();

    }
```

Gambar 5.11. Service Login

Antarmuka *login* pada Gambar 5.9. berfungsi untuk melakukan autentikasi terhadap pengguna yang akan memasuki sistem. Pengguna perlu memasukkan `username` dan `password` pada *input fields* dan pengguna yang sudah terautentikasi dapat mengakses layanan pada sistem. Gambar 5.10. dan Gambar 5.11.

merupakan potongan *code* yang menangani *login* pada sistem. *Controller login* digunakan untuk mengarahkan *api path* `/api/login` kepada *service login* `Response loginUsernameAndPassword(String username, String password)` untuk selanjutnya akan dilakukan pengecekan kepada basis data. `userRepository.loginUserWithUsernameAndPassword(username, password);` akan menggunakan *parameter* `username` dan `password` untuk mencari data pada basis data. Jika data ditemukan maka pesan respon yang dikembalikan sistem adalah ok dan jika tidak ditemukan, maka pesan respon adalah *failed*.

5.3.2. Third Party Logistic



Gambar 5.12. Antarmuka Third Party Logistic

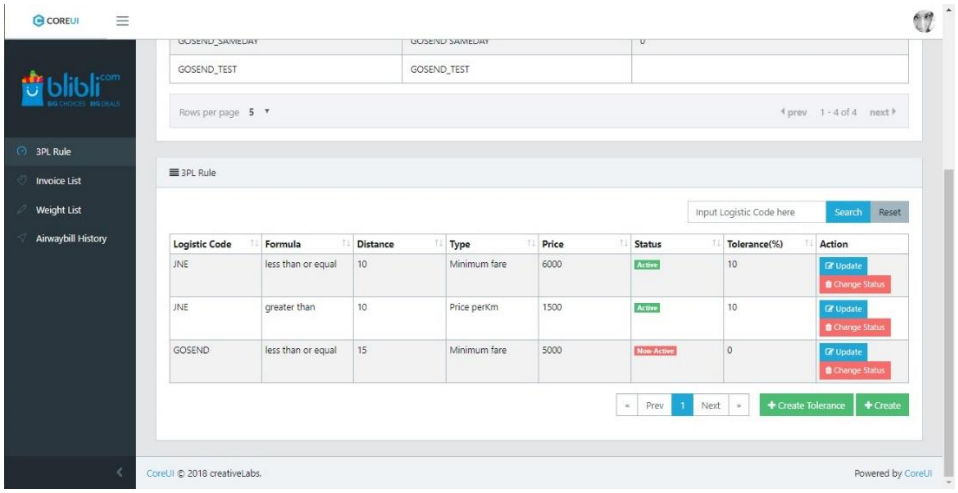
```
methods: {
  get3rdPl: function () {
    const URL = 'http://localhost:80/dummyJson/getAllProviders.json'
    axios.get(URL)
      .then(response => {
        this.rows = response.data.content
      })
      .catch(error => {
        alert(error, '', 'error')
      })
  },
```

Gambar 5.13. API JSON dummy local

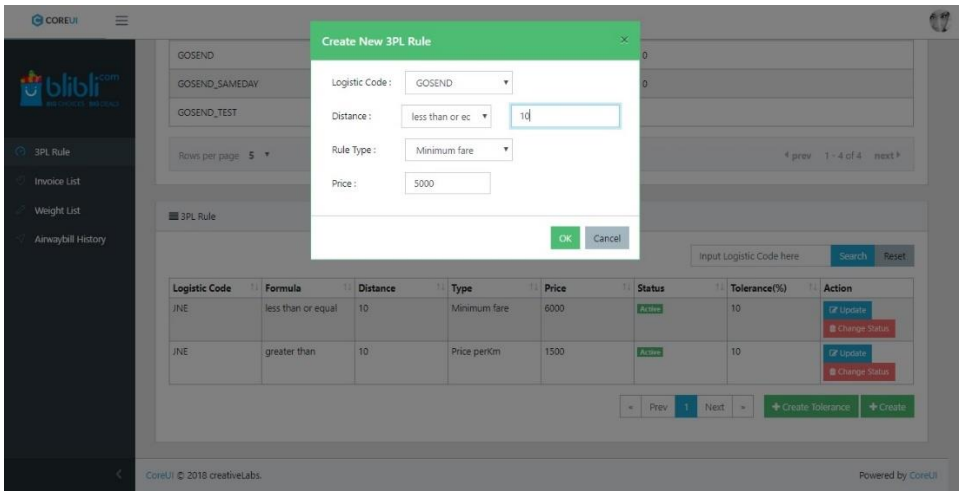
Antarmuka *third party logistic* pada Gambar 5.12. berfungsi untuk menampilkan daftar *third party logistic* pada sistem. Untuk mendapatkan data, sistem membutuhkan API lokal dari perusahaan yang menampilkan seluruh daftar *third party logistic* yang tersedia. Oleh karena keterbatasan akses API lokal maka dibuat suatu *dummy* JSON yang berisi daftar *third party logistic*. Untuk pengaksesan JSON tersebut, *front-end* hanya perlu mengambil dari *localhost* melalui fungsi `get3rdPl()` pada API

`'http://localhost:80/dummyJson/getAllProviders.json'` tanpa bantuan dari `back-end`. Contoh pengaksesan dapat dilihat pada Gambar 5.13.

5.3.3. Rule



Gambar 5.14. Antarmuka Rule



Gambar 5.15. Antarmuka Formulir Rule

```
@RestController
@RequestMapping("/api/rule")
public class RuleController {

    @Autowired
    private RuleService ruleService;

    @{...}
    public Response index(){
        return ruleService.getAll();
    }

    @{...}
    public Response store(@Valid @RequestBody RuleRequest request){
        return ruleService.store(request);
    }

    @{...}
    public Response update(@PathVariable String id, @Valid @RequestBody RuleRequest request){
        return ruleService.update(request,id);
    }
```

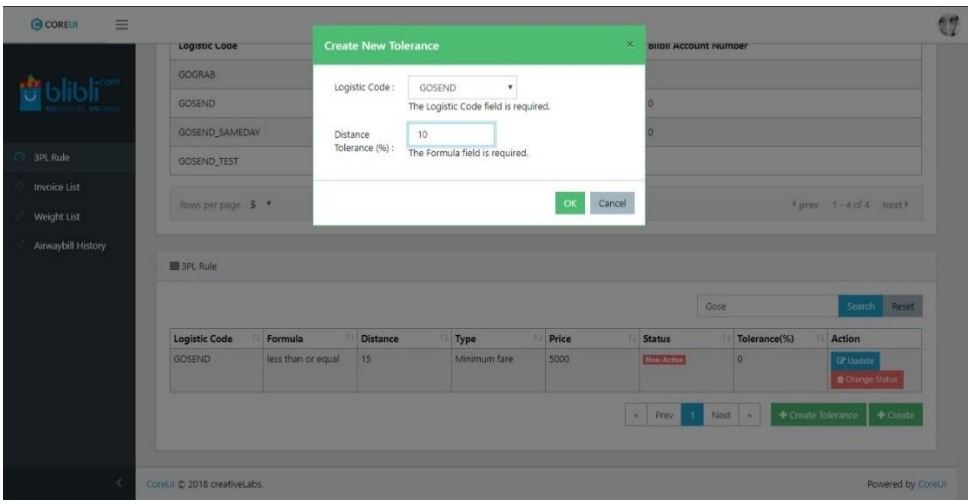
Gambar 5.16. Controller Rule

```
public Response getAll(){
    return Response.builder()
            .message("OK")
            .data(ruleRepository.findAll())
            .build();
}

@Override
public Response update(RuleRequest ruleRequest, String id){
    if (ruleRepository.exists(id)) {
        ruleRepository.updateToleranceThirdPartyLogistic(
                ruleRequest.getLogisticCode(),ruleRequest.getDistanceTolerance());
        Rule rule = ruleRepository.findOne(id);
        rule.setDistance(ruleRequest.getDistance());
        rule.setPrice(ruleRequest.getPrice());
        rule.setRuleType(ruleRequest.getRuleType());
        rule.setFormula(ruleRequest.getFormula());
        rule.setLogisticCode(ruleRequest.getLogisticCode());
        return Response.builder()
                .message("Rule Successfully Updated!")
                .data(ruleRepository.save(rule))
                .build();
    } else {
        return Response.builder()
                .message("Rule Not Found!!")
                .build();
    }
}
```

Gambar 5.17. Service Rule

Antarmuka *rule* pada Gambar 5.14. dan Gambar 5.15. berfungsi untuk melakukan kelola data *rule* pada suatu *third party logistic* seperti melihat informasi daftar *rule*, membuat, mengubah, mengubah status, dan melakukan pencarian data. Data yang ditampilkan pada tabel dalam paginasi lima data *rule* per halaman. Gambar 5.16. dan Gambar 5.17. merupakan contoh potongan *code* yang menangani fitur *rule* pada sistem. *Controller rule* digunakan untuk mengarahkan *api path* `@RequestMapping("/api/rule")` sesuai dengan *method* yang berikan seperti POST, PUT, GET, dan DELETE kepada *service rule*. Pada *service rule* akan dilakukan logika bisnis terhadap fungsi-fungsi yang dibutuhkan pada *service rule*. Untuk melakukan akses kepada basis data, *service rule* akan menggunakan `ruleRepository`, untuk mengambil seluruh data *rule* maka akan menggunakan `ruleRepository.findAll()` dan mengirimkan respon berupa JSON dengan pesan ok.

5.3.4. Distance Tolerance



Gambar 5.18. Antarmuka Distance Tolerance

```
@RequestMapping(
        value = "/updateTolerance",
        method = RequestMethod.PUT
)
public Response updateTolerance(@Valid @RequestBody
    ToleranceThirdPartyLogisticRequest tolerance){
    return ruleService.updateToleranceThirdPartyLogistic(tolerance);
}
```

Gambar 5.19. Controller Distance Tolerance

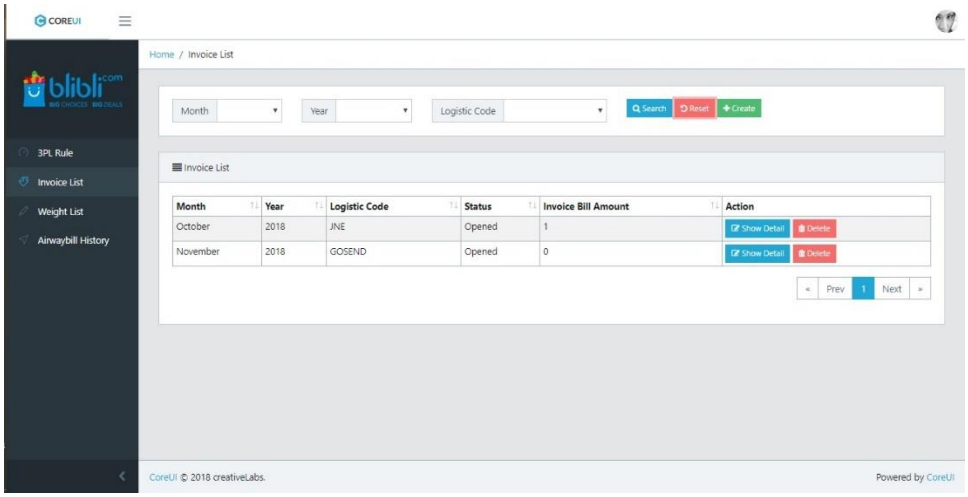
```
public Response updateToleranceThirdPartyLogistic(ToleranceThirdPartyLogisticRequest tolerance){
    try {
        Rule rule = ruleRepository.findRuleByLogisticCode(tolerance.getLogisticCode());
        if (rule != null && rule.getDistanceTolerance() == 0) {
            ruleRepository.updateToleranceThirdPartyLogistic(tolerance.getLogisticCode(),
                tolerance.getDistanceTolerance());
            return Response.builder()
                    .message("Add Tolerance for " + tolerance.getLogisticCode() + " Success!")
                    .data(ruleRepository.findAll())
                    .build();
        }
        else if(rule.getDistanceTolerance() != 0){
            return Response.builder()
                    .message("Failed, You Can Only Create 1 Tolerance For This Logistic Code")
                    .build();
        }
    }catch (Exception e) {
        logger.info(e.toString());
    }
    return Response.builder()
            .message("Failed, Please Check Your Rule. ")
            .build();
}
```

Gambar 5.20. Service Distance Tolerance

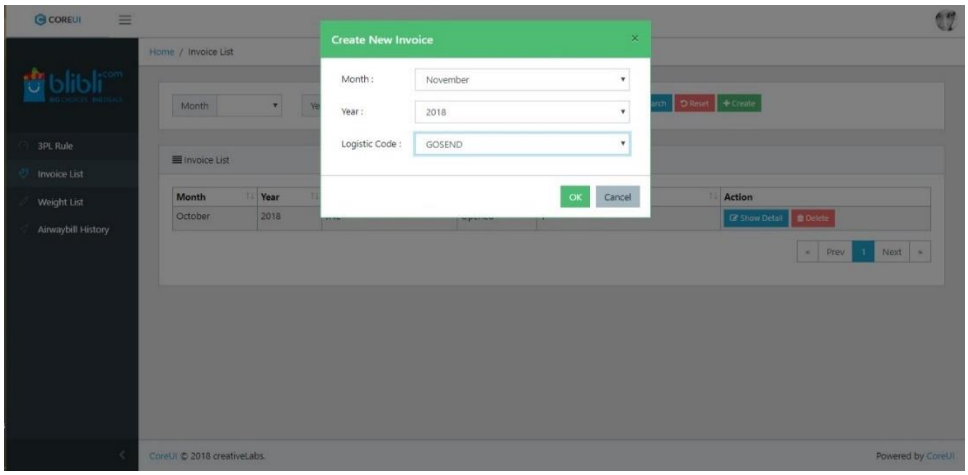
Antarmuka *distance tolerance* pada Gambar 5.18. berfungsi untuk melakukan kelola data *distance tolerance* pada suatu *third party logistic*. *Distance tolerance* merupakan salah satu *field* yang terdapat pada model *rule* sehingga data *distance tolerance* ditampilkan dalam tabel *rule*. *Controller distance tolerance* digunakan untuk mengarahkan *api path* kepada *service distance tolerance* untuk

selanjutnya akan dilakukan logika bisnis terhadap fungsi `updateToleranceThirdPartyLogistic.ruleRepository.findRuleByLogisticCode(tolerance.getLogisticCode());` digunakan untuk mencari *rule* berdasarkan *logistic code* yang data *distance tolerance* akan diubah. Jika data ditemukan dan data *distance tolerance* masih nol, maka akan dilakukan ubah data *distance tolerance* sesuai dengan masukan dari pengguna. Jika *distance tolerance* tidak nol, maka sistem akan mengembalikan pesan respon bahwa pengguna hanya dapat membuat *distance tolerance* sebanyak satu kali.

5.3.5. Invoice



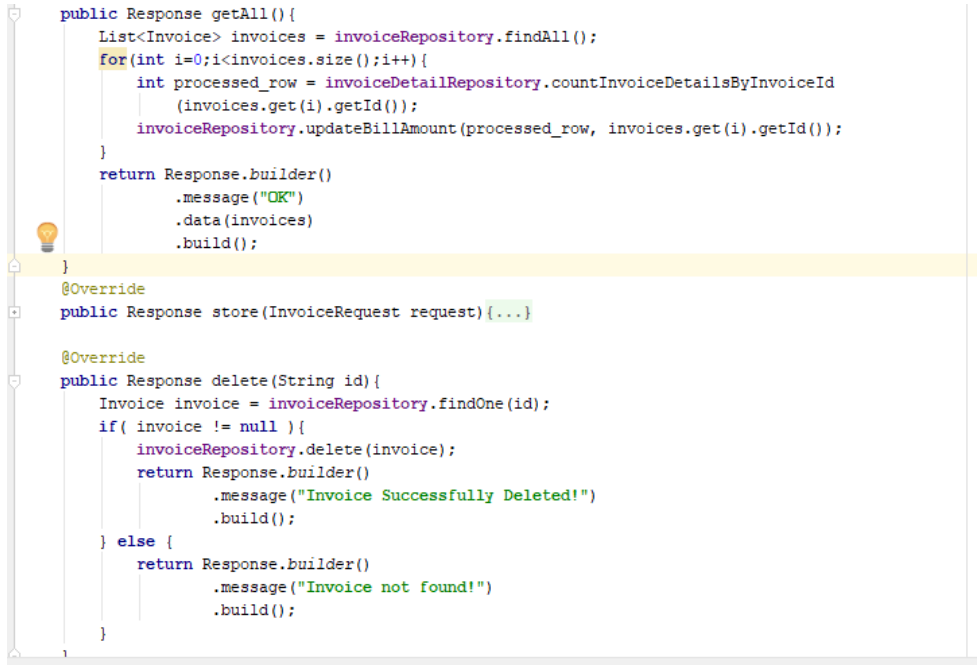
Gambar 5.21. Antarmuka Invoice



Gambar 5.22. Antarmuka Formulir Invoice

```
@RestController
@RequestMapping("/api/invoice")
public class InvoiceController {

    @Autowired
    private InvoiceService invoiceService;

    @{...}
    public Response index(){
        return invoiceService.getAll();
    }

    @{...}
    public Response store(@Valid @RequestBody InvoiceRequest request){
        return invoiceService.store(request);
    }

    @{...}
    public Response delete(@PathVariable String id){
        return invoiceService.delete(id);
    }
```

Gambar 5.23. Controller Invoice



```
    public Response getAll(){
        List<Invoice> invoices = invoiceRepository.findAll();
        for(int i=0;i<invoices.size();i++){
            int processed_row = invoiceDetailRepository.countInvoiceDetailsByInvoiceId
                (invoices.get(i).getId());
            invoiceRepository.updateBillAmount(processed_row, invoices.get(i).getId());
        }
        return Response.builder()
                .message("OK")
                .data(invoices)
                .build();
    }
    @Override
    public Response store(InvoiceRequest request){...}

    @Override
    public Response delete(String id){
        Invoice invoice = invoiceRepository.findOne(id);
        if( invoice != null ){
            invoiceRepository.delete(invoice);
            return Response.builder()
                    .message("Invoice Successfully Deleted!")
                    .build();
        } else {
            return Response.builder()
                    .message("Invoice not found!")
                    .build();
        }
    }
```

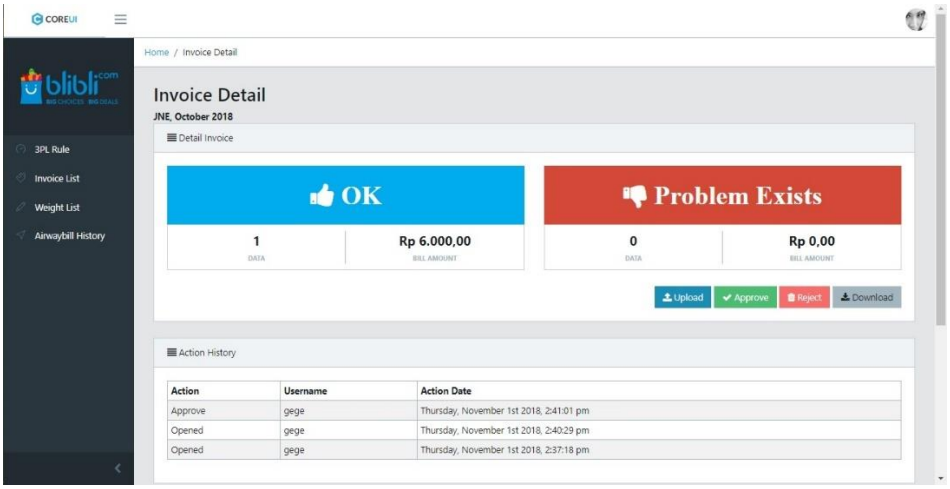
Gambar 5.24. Service Invoice

Antarmuka *invoice* pada Gambar 5.21. dan Gambar 5.22. berfungsi untuk melakukan kelola data *invoice* pada suatu *third party logistic* seperti melihat informasi daftar *invoice*, membuat, menghapus, dan melakukan pencarian data. Data yang ditampilkan pada tabel dalam paginasi lima data *invoice* per halaman. Gambar 5.23. dan Gambar 5.24. merupakan potongan *code* yang menangani fitur *invoice* pada sistem. *Controller invoice* digunakan untuk mengarahkan *api path* `@RequestMapping("/api/rule")` sesuai dengan *method* yang berikan seperti POST, PUT, GET, dan DELETE kepada *service invoice*. Pada *service* akan dilakukan logika bisnis terhadap fungsi-fungsi yang dibutuhkan pada fitur *invoice*.

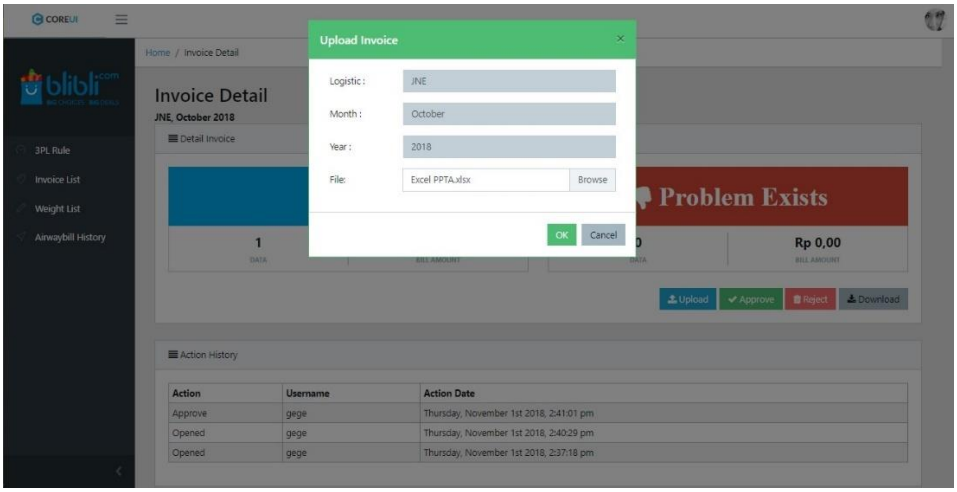
Untuk melakukan akses kepada basis data, *service invoice* akan menggunakan `invoiceRepository`. Untuk menghapus suatu data maka akan

menggunakan `invoiceRepository.findOne()`, dimana sistem akan mencari data *invoice* berdasarkan *id*. Jika data ditemukan maka data akan dihapus menggunakan `invoiceRepository.delete()`.

5.3.6. Invoice Detail



Gambar 5.25. Antarmuka Invoice Detail



Gambar 5.26. Antarmuka Unggah Berkas Invoice

```
@Autowired
private InvoiceDetailService invoiceDetailService;

@{...}
public Response uploadFile(@RequestParam("file") MultipartFile uploadFile,
                           @PathVariable String id) {...}

@{...}
public Response getCountAwbSuccess(@PathVariable String id){...}

@{...}
public Response getCountAwbProblemExist(@PathVariable String id){...}

@{...}
public Response getSumBillAmountProblemExist(@PathVariable String id){...}

@{...}
public Response getSumBillAmountSuccess(@PathVariable String id){...}

@{...}
public Response getHeaderInvoice(@PathVariable String id){...}

@{...}
public byte[] invoiceExcelExport(@PathVariable String id) throws IOException {...}
```

Gambar 5.27. Controller Invoice Detail

```
@Override
public Response getSumBillAmountSuccess(String id) {

    return Response.builder()
            .message("OK")
            .data(invoiceDetailRepository.getSumBillAmountSuccess(id))
            .build();
}

@Override
public Response getSumBillAmountProblemExist(String id) {

    return Response.builder()
            .message("OK")
            .data(invoiceDetailRepository.getSumBillAmountProblemExist(id))
            .build();
}
```

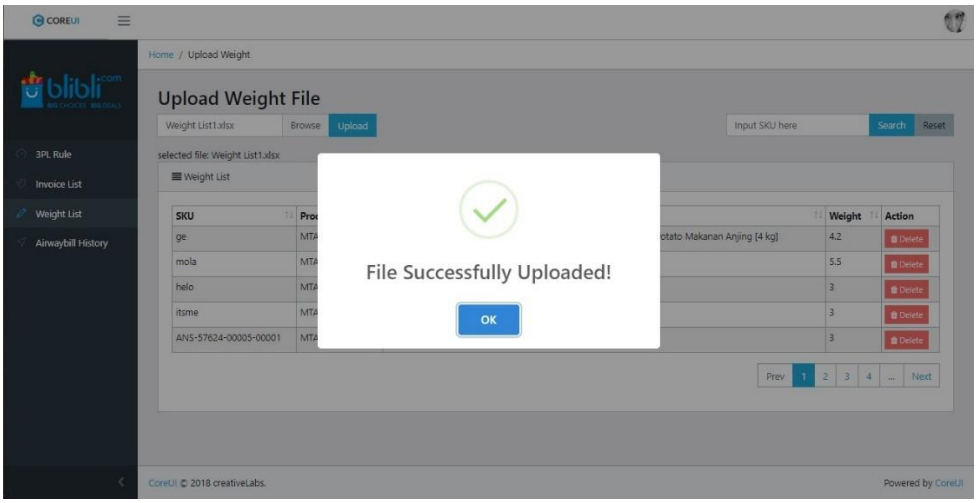
Gambar 5.28. Service Invoice Detail

Antarmuka *invoice detail* pada Gambar 5.25. dan Gambar 5.26. berfungsi untuk melakukan kelola data *invoice detail* pada suatu *invoice* seperti melihat informasi berupa dashboard, mengunggah berkas (.xlsx), mengunduh berkas, dan melakukan perubahan aksi yaitu *reject* dan *approve*. Gambar 5.27. dan Gambar 5.28. merupakan potongan *code* yang menangani fitur *invoice detail* pada sistem. *Controller invoice detail* digunakan untuk mengarahkan *api path* kepada *service invoice detail*, untuk selanjutnya akan dilakukan logika bisnis terhadap fungsi-fungsi yang dibutuhkan pada fitur *invoice detail*.

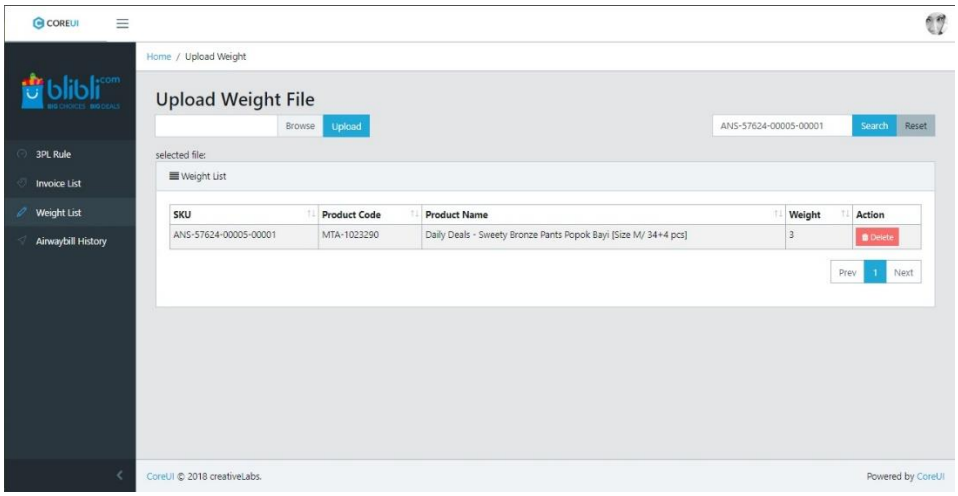
Untuk melakukan akses kepada basis data, *service invoice detail* akan menggunakan `invoiceDetailRepository`. Fungsi untuk melakukan melakukan pengambilan data dengan status berhasil berdasarkan *id*. Proses bisnis yang dilakukan akan membutuhkan data dari basis data, sehingga menggunakan

`invoiceDetailRepository.getSumBillAmountSuccess(id)` dan mengirimkan respon berupa JSON dengan pesan ok.

5.3.7. Weight List



Gambar 5.29. Antarmuka Unggah Berkas Weight List



Gambar 5.30. Antarmuka Weight List

```
@RestController
@RequestMapping("/api/weightList")
public class WeightListController {

    @Autowired
    private SkuService skuService;

    @RequestMapping(method = RequestMethod.POST)
    public Response uploadFile(@RequestParam("file") MultipartFile uploadfile) {...}

    @{...}
    public Response getAll() {...}

    @{...}
    public Response getAllByKey(@RequestParam("key") String key) {...}

    @{...}
    public Response delete(@PathVariable String id){...}
}
```

Gambar 5.31. Controller Weight List

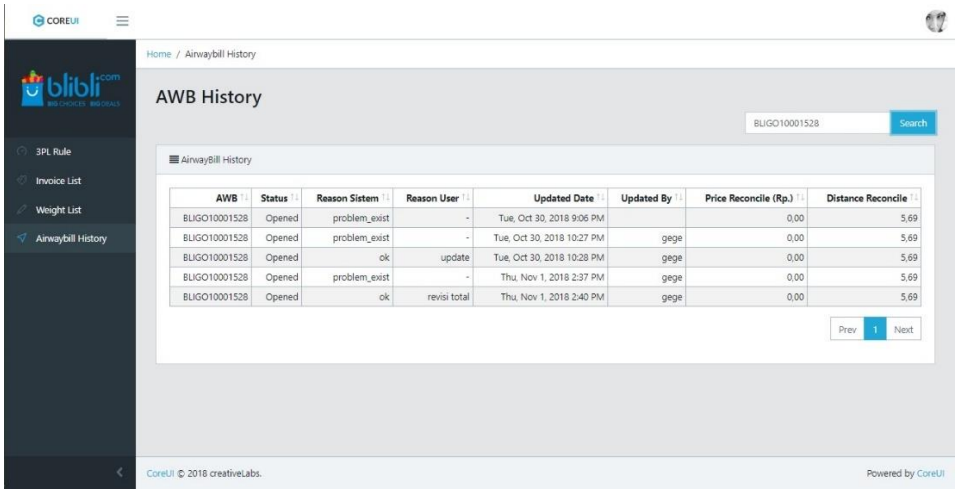
```java
@Override
public Response uploadFile(MultipartFile uploadFile){
    if (uploadFile.isEmpty()) {
        return Response.builder()
                .message("Select File!")
                .build();
    }
    try {
        saveUploadedFiles(Arrays.asList(uploadFile));
        testJobLauncer();
    } catch (IOException e) {
        return Response.builder()
                .message("Failed!")
                .build();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return Response.builder()
            .message("Succesfully Upload!")
            .build();
}

@Override
public void saveUploadedFiles(List<MultipartFile> files) throws Exception {...}
```

Gambar 5.32. Service Weight List

Antarmuka *weight list* pada Gambar 5.29. dan Gambar 5.30. berfungsi untuk melakukan kelola data *weight list*, seperti mengunggah berkas (.xlsx), melihat informasi *weight list*, menghapus, dan melakukan pencarian. Gambar 5.31. dan Gambar 5.32. merupakan potongan *code* yang menangani fitur *weight list* pada sistem. *Controller weight list* digunakan untuk mengarahkan *api path* `@RequestMapping("/api/weightList")`, sesuai dengan *method* yang berikan kepada *service weight list*. Pada *service* akan dilakukan logika bisnis terhadap fungsi-fungsi yang dibutuhkan untuk fitur *weight list*.

Untuk melakukan akses kepada basis data, *service* akan menggunakan `skuRepository`. Terdapat fungsi `Response uploadFile(MultipartFile uploadFile)` untuk melakukan unggah berkas. Proses bisnis yang terjadi adalah melakukan pengecekan terhadap berkas(.xlsx) dan jika berkas tidak kosong, maka sistem akan memanggil fungsi `saveUploadedFiles(Arrays.asList(uploadFile))`. Fungsi tersebut digunakan untuk menyimpan berkas ke dalam *server*, yang selanjutnya memanggil fungsi `testJobLauncer()` untuk menjalankan *batch processing*.

5.3.8. AirwayBill History



Gambar 5.33. AirwayBill History

```
@RestController
@RequestMapping("/api/airwayBillHistory")
public class AirwayBillHistoryController {

    @Autowired
    private AirwayBillHistoryService airwayBillHistoryService;

    @RequestMapping(
            value = "/search",
            method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON_VALUE
    )
    public Response getAllByKey(@RequestParam("key") String key) {

        return airwayBillHistoryService.getAllByKey(key);
    }
}
```

Gambar 5.34. Controller AirwayBill History

```
    @Autowired
    private AirwayBillHistoryRepository airwayBillHistoryRepository;

    @Override
    public Response getAllByKey(String key){...}

    @Override
    public Response store(InvoiceDetail invoiceDetail, ActionHistory actionHistory){
        AirwayBillHistory airwayBillHistory = AirwayBillHistory.builder()
                .id(UUID.randomUUID().toString())
                .awbNo(invoiceDetail.getId())
                .reasonSistem(invoiceDetail.getStatusFinal())
                .reasonUser(invoiceDetail.getReasonUser())
                .status(actionHistory.getAction())
                .statusDistance(invoiceDetail.getStatusDistance())
                .statusPrice(invoiceDetail.getStatusPrice())
                .updatedBy(actionHistory.getUsername())
                .updatedDate(actionHistory.getActionDate())
                .distanceReconcile(invoiceDetail.getDistanceGoogleMapsApi())
                .build();
        return Response.builder()
                .message("OK")
                .data(airwayBillHistoryRepository.save(airwayBillHistory))
                .build();
    }
}
```

Gambar 5.35. Service AirwayBill History

Antarmuka *airwaybill history* pada Gambar 5.33. berfungsi untuk melihat riwayat dari suatu *airwaybill* yang dimasukkan pada *input search field*. Gambar

5.34. dan Gambar 5.35. merupakan potongan *code* yang menangani fitur *airwaybill history* pada sistem. *Controller airwaybill history* digunakan untuk mengarahkan *api path* `@RequestMapping("/api/airwayBillHistory")` dengan method GET dan POST kepada *service airwaybill history*. Pada *service* akan dilakukan logika bisnis terhadap fungsi-fungsi yang dibutuhkan fitur *airwaybill history*.

Service akan membutuhkan akses kepada basis data, sehingga menggunakan `airwayBillHistoryRepository`. Potongan code untuk melakukan penyimpanan terhadap data *airwaybill history*. `Response store(InvoiceDetail invoiceDetail, ActionHistory actionHistory)` merupakan fungsi yang memiliki dua parameter. `airwayBillHistoryRepository.save (airwayBillHistory)` merupakan fungsi yang akan mengeksekusi query, untuk melakukan penyimpanan ke dalam basis data.

5.3.9. Batch Configuration Code

Batch Configuration merupakan bagian utama untuk melakukan konfigurasi setiap komponen penyusun *batch processing* pada Spring Batch yang terdapat pada contoh Gambar 5.1. Berikut potongan *code* inti yang terdapat pada *Batch Configuration*:

1. Method *Job*

```
public Job importInvoiceDetailJob(
String originName, String invoiceId) throws Exception {
  return    jobBuilderFactory.get("importInvoiceDetailJob
")
                .incrementer(new RunIdIncrementer())
                .flow(stepInvoice(originName,invoiceId))
                .end()
                .build();}
```

Code di atas merupakan komponen *Job* yang akan mengeksekusi *batch processing*. Nama *Job* dideklarasikan sebagai `importInvoiceDetailJob`, yang akan berisi kumpulan dari *step* yang akan dijalankan oleh `JobLauncher`. *Method* ini akan memiliki dua parameter yaitu `originName` dan `invoiceId`. Kedua parameter akan digunakan untuk membaca berkas yang telah diunggah ke dalam *server*, melalui `stepInvoice(originName,invoiceId)`. Untuk setiap *Job* akan memiliki suatu *id* unik, yang akan membedakan antara setiap *Job* yang sudah dilakukan atau akan dilakukan. *Id* akan terbentuk secara otomatis dengan `incrementer(new RunIdIncrementer())`.

2. Method *Job Launcher*

```
public void JobInvoiceLauncer() throws Exception {
   jobLauncher
       .run(invoiceBatchConfiguration
          .importInvoiceDetailJob
              (originName,invoiceId), newExecution());
}
```

Code di atas merupakan *Job Launcher*, *method* ini akan dipanggil ketika *batch processing* akan dijalankan. Dalam menjalankan sebuah *batch job*, minimal ada dua hal yang dibutuhkan oleh Spring Batch yaitu *Job* dan *Job Launcher*. *Job* akan berisi apa yang akan dijalankan, sedangkan *Job Launcher* akan mengeksekusi *Job* yang telah ditentukan. Dalam pengimplementasian, *Job Launcher* akan dipanggil ketika pengguna melakukan pengunggahan berkas.

3. Method Step

```
public Step stepInvoice(String originName, String
invoiceId) throws Exception {
    return stepBuilderFactory.get("stepInvoice")
         .<InvoiceDetail,InvoiceDetail>chunk(200)
          .reader(reader(originName, invoiceId))
          .processor(processor())
          .writer(writers())
          .build();
}
```

Code di atas merupakan komponen *step* yang berisi tahap-tahap seperti membaca data, melakukan proses, dan menuliskan data ke dalam basis data. `reader(reader(originName, invoiceId)` merupakan proses pembacaan data pada suatu berkas. `processor(processor())` merupakan proses yang akan dilalui oleh setiap baris dalam berkas, seperti melakukan kalkulasi dan pencarian melalui Google Maps API. `writer(writers())` merupakan proses penulisan kembali ke dalam basis data. *Chunk* terdapat dalam *method step*, yang merupakan sebuah pendekatan dalam membagi data ke dalam sebuah *batch* dengan jumlah tertentu. Hasil yang diharapkan dalam penggunaan *chunk* adalah data yang dimasukkan ke dalam sistem tidak langsung dimasukkan semua, namun dipecah menjadi bagian yang lebih kecil.

5.4. Pengujian Sistem

Pengujian perangkat lunak SIJAHA adalah menggunakan *unit test* untuk menguji fungsionalitas proses bisnis, yang terdapat pada *back-end* dengan Junit 4 dan Mockito sebagai *dependency* tambahan. Pengujian pada perangkat lunak

adalah semua kelas pada *package controller* dan *service implementation*, sesuai dengan kebutuhan perusahaan. Pengujian yang dilakukan adalah *Line Coverage* dan *Method Coverage*. Pengujian dikatakan berhasil jika *minimum coverage* adalah 90%. Terdapat perbedaan dalam melakukan pengujian antara *package controller* dan *service*. *Package controller* akan dilakukan pengujian pada hasil JSON yang dikembalikan oleh suatu *path* API. *Package service* berisi proses bisnis, sehingga akan dilakukan pengujian terhadap setiap fungsi-fungsi yang digunakan.

Pengujian dengan *unit test* akan terdiri dari tiga bagian utama pada setiap kelasnya yaitu *Before*, *Method*, dan *After*. *Before* merupakan tahap dimana *unit test* akan melakukan inisialisasi, berupa mendeklarasikan isi dari suatu objek yang akan di-*mock*. *Method* merupakan fungsi yang akan diuji dan *After* merupakan tahap yang dijalankan terakhir. *After* berfungsi untuk memastikan bahwa tidak ada lagi interaksi pada suatu objek atau variabel. Untuk *Before* dan *After* secara keseluruhan memiliki struktur yang sama pada setiap kelas *unit test*, maka akan diberikan contoh potongan *code* seperti berikut:

```
@Before
public void init(){
   MockitoAnnotations.initMocks(this);
   mockMvc = MockMvcBuilders
   .standaloneSetup(actionHistoryController)
   .build();
}
```

Pada method `init()` dimana memiliki anotasi `@Before` akan dijalankan pertama dan berguna untuk menginisialisasi kelas *unit test*. `standaloneSetup` berfungsi untuk menerima objek sebagai parameter.

```
@After
public void tearDown(){
   verifyNoMoreInteractions(actionHistoryService);
}
```

Pada *method* `tearDown()` dimana memiliki anotasi `@After` akan dijalankan terakhir atau setelah method dari fungsi dengan anotasi `@Test` dijalankan. *Method* ini berguna untuk memastikan kepada Junit, bahwa objek yang telah di-*mock* sudah dipanggil.

Berikut potongan *code* pada `controller action history`:

```
@Test
public void testGetAllByInvoiceId() throws Exception{
    List<ActionHistory> actionHistories = Arrays.asList(
            new ActionHistory(actionHistoryId,action,username,actionDate,invoice));
    response.setData(actionHistories);
    when(actionHistoryService.getAllByInvoiceId(actionHistoryId)).thenReturn(response);
    this.mockMvc.perform(get( urlTemplate: "/api/actionHistory/01-future-test"))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$.data",hasSize(1)))
            .andExpect(jsonPath( expression: "$.data[0].actionHistoryId", equalTo(actionHistoryId)))
            .andExpect(jsonPath( expression: "$.data[0].action", equalTo(action)))
            .andExpect(jsonPath( expression: "$.data[0].username", equalTo(username)))
            .andExpect(jsonPath( expression: "$.data[0].actionDate", equalTo(actionDateConvert)))
            .andExpect(jsonPath( expression: "$.data[0].invoice", equalTo(invoice)));
    verify(actionHistoryService, times( wantedNumberOfInvocations: 1)).getAllByInvoiceId(actionHistoryId);
}
@Test
public void testApproveSuccess() throws Exception {
    actionHistory.setActionHistoryId(actionHistoryId);
    actionHistory.setAction(action);
    actionHistory.setActionDate(actionDate);
    actionHistory.setUsername(username);
    actionHistory.setInvoice(invoice);
    mockMvc.perform(
            post( urlTemplate: "/api/actionHistory/approve/"+actionHistoryId)
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk());
    verify(actionHistoryService, times( wantedNumberOfInvocations: 1))
            .store(actionHistory.getActionHistoryId(),actionHistory.getAction(),actionHistory.getUsername());
}
```

Gambar 5.36. Unit test controller action history

Gambar 5.36. merupakan *unit test* pada `controller action history` dengan menguji JSON pada suatu API. `testGetAllByInvoiceId` merupakan API dengan *method* GET, sehingga perlu dilakukan *unit test* untuk memastikan JSON yang dikembalikan oleh API sesuai dengan data yang seharusnya. `testApproveSuccess` merupakan API dengan *method* POST, berbeda dengan *method* GET, *unit test* diperlukan untuk memastikan data berhasil disimpan melalui *response code* 200 OK.

Berikut potongan *code* pada `controller airwaybill history`:

```
@Test
public void testGetAllByKeyAirwayBillHistorySuccess() throws Exception{

    List<AirwayBillHistory> airwayBillHistories = Arrays.asList(
            new AirwayBillHistory(id,awbNo,status,reasonSistem,reasonUser,
                    updateDate,updateBy,statusPrice,statusDistance,hargaReconcile,distanceReconcile));

    response.setData(airwayBillHistories);
    when(airwayBillHistoryService.getAllByKey(key)).thenReturn(response);

    this.mockMvc.perform(get( urlTemplate: "/api/airwayBillHistory/search").param( name: "key",key))
            .andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8_VALUE))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$.data",hasSize(1)))
            .andExpect(jsonPath( expression: "$.data[0].id", equalTo(airwayBillHistories.get(0).getId())))
            .andExpect(jsonPath( expression: "$.data[0].awbNo", equalTo(airwayBillHistories.get(0).getAwbNo())))
            .andExpect(jsonPath( expression: "$.data[0].status", equalTo(airwayBillHistories.get(0).getStatus())))
            .andExpect(jsonPath( expression: "$.data[0].reasonSistem", equalTo(airwayBillHistories.get(0).getReasonSistem())))
            .andExpect(jsonPath( expression: "$.data[0].reasonUser", equalTo(airwayBillHistories.get(0).getReasonUser())))
            .andExpect(jsonPath( expression: "$.data[0].updatedDate", equalTo(airwayBillHistories.get(0).getUpdatedDate().getTime())))
            .andExpect(jsonPath( expression: "$.data[0].updatedBy", equalTo(airwayBillHistories.get(0).getUpdatedBy())))
            .andExpect(jsonPath( expression: "$.data[0].statusPrice", equalTo(airwayBillHistories.get(0).getStatusPrice())))
            .andExpect(jsonPath( expression: "$.data[0].statusDistance", equalTo(airwayBillHistories.get(0).getStatusDistance())))
            .andExpect(jsonPath( expression: "$.data[0].hargaReconcile", equalTo(airwayBillHistories.get(0).getHargaReconcile())))
            .andExpect(jsonPath( expression: "$.data[0].distanceReconcile", equalTo(airwayBillHistories.get(0).getDistanceReconcile())));

    verify(airwayBillHistoryService, times( wantedNumberOfInvocations: 1)).getAllByKey(key);
}
```

Gambar 5.37. Unit test controller airwaybill history

Gambar 5.37. merupakan *unit test* pada `controller airwaybill history` dengan menguji JSON pada suatu API. `testGetAllByKeyAirwayBillHistorySuccess()` merupakan *unit test* untuk API dengan *method* GET. *Unit test* perlu dilakukan untuk memastikan JSON yang dikembalikan oleh API `get("/api/actionHistory/01-future-test")` sesuai dengan data `airwaybill` yang seharusnya.

`when(airwayBillHistoryService.getAllByKey(key)).thenReturn (response)` ini merupakan *method* untuk mengembalikan nilai, ketika suatu fungsi yang telah di-*mock* dipanggil. Contoh potongan *code* tersebut mengembalikan objek `Response` ketika fungsi `airwayBillHistoryService.getAllByKey(key)` dipanggil. `verify(airwayBillHistoryService,times(1)).getAllByKey (key)` *method* ini berfungsi untuk memastikan bahwa fungsi `airwayBillHistoryService.getAllByKey(key)` sudah terpanggil, setidaknya satu kali dalam *unit test*.

Berikut potongan *code* pada `controller action history`:

```
@Test
public void testApproveSuccess() throws Exception {
    actionHistory.setActionHistoryId(actionHistoryId);
    actionHistory.setAction(action);
    actionHistory.setActionDate(actionDate);
    actionHistory.setUsername(username);
    actionHistory.setInvoice(invoice);
    mockMvc.perform(
            post( urlTemplate: "/api/actionHistory/approve/"+actionHistoryId)
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk());
    verify(actionHistoryService, times( wantedNumberOfInvocations: 1))
            .store(actionHistory.getActionHistoryId(),actionHistory.getAction(),actionHistory.getUsername());
}
```

Gambar 5.38. Unit test controller action history

Gambar 5.38. merupakan unit test pada `controller action history` dengan menguji JSON pada API. `testApproveSuccess()` merupakan *unit test* untuk API dengan *method* POST. *Unit test* perlu dilakukan untuk memastikan data JSON yang diberikan pada API `post("/api/actionHistory/approve/" +actionHistoryId`, berhasil disimpan ke dalam sistem. `andExpect(status().isOk())` berfungsi untuk melakukan pengecekan bahwa status yang dikembalikan oleh sistem adalah 202 OK. `verify()`merupakan *method* yang berfungsi untuk memastikan bahwa fungsi `actionHistoryService.store(?,?,?)`, sudah terpanggil setidaknya satu kali dalam *unit test*.

Berikut potongan *code* pada `controller invoice`:

```
@Test
public void testDeleteSuccess() throws Exception {
    mockMvc.perform( delete( urlTemplate: "/api/invoice/"+invoice.getId()))
            .andExpect(status().isOk());
    verify(invoiceService, times( wantedNumberOfInvocations: 1)).delete(invoice.getId());
    verifyNoMoreInteractions(invoiceService);
}
}
```

Gambar 5.39. Unit test controller invoice

Gambar 5.39. merupakan *unit test* pada `controller invoice` dengan menguji JSON pada API. `testDeleteSuccess()` merupakan unit test untuk API dengan *method* DELETE. *Unit test* perlu dilakukan untuk memastikan data yang akan dihapus melalui API `delete("/api/invoice/"+invoice.getId())` berhasil dihapus dari dalam sistem. `andExpect(status().isOk())` berfungsi untuk melakukan pengecekan, bahwa status yang dikembalikan oleh sistem adalah 202 OK. `verify(invoiceService, times(1)).delete(invoice.getId())` merupakan *method*, yang berfungsi untuk memastikan bahwa fungsi `invoiceService.delete(invoice.getId())` sudah terpanggil setidaknya satu kali dalam *unit test*.

Berikut potongan *code* pada `controller rule`:

```
@Test
public void testUpdateRuleSuccess() throws Exception {

    mockMvc.perform(
            put( urlTemplate: "/api/rule/{id}", rulereq.getLogisticCode())
                .contentType(MediaType.APPLICATION_JSON)
                .content(mapper.writeValueAsString(rulereq)))
                .andExpect(status().isOk());

    verify(ruleService, times( wantedNumberOfInvocations: 1)).update(rulereq,rulereq.getLogisticCode());
}
```

Gambar 5.40. Unit test controller rule

Gambar 5.40. merupakan *unit test* pada `controller rule` dengan menguji JSON pada API. `testUpdateRuleSuccess()` merupakan *unit test* untuk API dengan *method* PUT. *Unit test* perlu dilakukan untuk memastikan data yang akan diubah melalui API `put("/api/rule/{id}",rulereq.getLogisticCode())`, berhasil dilakukan. `andExpect(status().isOk())`berfungsi untuk melakukan pengecekan, bahwa status yang dikembalikan oleh sistem adalah 202 OK. *Method* `verify(ruleService,times(1)).update(rulereq,rulereq.getLogisticCode()`, berfungsi untuk memastikan bahwa fungsi `ruleService.`

`update(rulereq,rulereq.getLogisticCode())`, sudah terpanggil setidaknya satu kali dalam *unit test*.

Berikut potongan *code* pada `controller weight list`:

```
@Test
public void testUploadSuccess() throws Exception {
    String UPLOADED_FOLDER = "D://weight list//";
    FileInputStream fis = new FileInputStream( s: UPLOADED_FOLDER+"Weight List1.xlsx");
    MockMultipartFile multipartFile = new MockMultipartFile( name: "file", fis);

    MediaType mediaType = new MediaType( type: "multipart",  subtype: "form-data");

    mockMvc.perform(
            MockMvcRequestBuilders.fileUpload( urlTemplate: "/api/weightList").file(multipartFile)
                .content(multipartFile.getBytes())
                .contentType(mediaType))
                .andExpect(status().isOk());
    verify(skuService, times( wantedNumberOfInvocations: 1)).uploadFile(multipartFile);
}
```

Gambar 5.41. Unit test controller weight list

Gambar 5.41. merupakan *unit test* pada `controller weight list` dengan menguji JSON pada API. `testUploadSuccess()` merupakan *unit test* untuk API unggah berkas(.xlsx) *weight list*. *Unit test* perlu dilakukan untuk memastikan berkas yang diunggah melalui API `"/api/weightList"` berhasil diunggah. `FileInputStream fis = new FileInputStream(UPLOADED_FOLDER+"Weight List1.xlsx")`, berfungsi untuk membuat berkas dengan mengambil dari `path` yang ada di dalam *server*. `MockMultipartFile multipartFile = new MockMultipartFile("file", fis)` akan melakukan *mock*, untuk `multipartfile` seperti berkas(.xlsx) yang diunggah. `verify(skuService, times(1)).uploadFile(multipartFile)` merupakan *method* yang berfungsi untuk memastikan, bahwa *method* `skuService.uploadFile(multipartFile)` sudah terpanggil setidaknya satu kali dalam *unit test*.

Berikut potongan *code* pada `service action history`:

```
@Test
public void testGetAllByInvoiceIdSuccess() throws Exception{

    response.setData(actionHistories);
    when(actionHistoryRepository.findAllByInvoiceId(invoiceId)).thenReturn(actionHistories);
    assertEquals(response,actionHistoryServiceImpl.getAllByInvoiceId(invoiceId));
    verify(actionHistoryRepository, times( wantedNumberOfInvocations: 1)).findAllByInvoiceId(invoiceId);
}
```

Gambar 5.42. Unit test service action history

Gambar 5.42. merupakan *unit test* pada `service action history` dengan menguji logika bisnis. `testGetAllByInvoiceIdSuccess()` merupakan

unit test untuk mengambil seluruh data `action history` pada suatu `invoice id`. *Unit test* perlu dilakukan untuk memastikan, data yang diambil sesuai dengan data seharusnya. `when(airwayBillHistoryService.getAllByKey(key)).thenReturn(response)` merupakan *method*, untuk mengembalikan nilai ketika suatu fungsi yang telah di-*mock* dipanggil.

Contoh potongan *code* tersebut mengembalikan objek `Response`, ketika fungsi `actionHistoryRepository.findAllByInvoiceId(invoiceId)` dipanggil. Jika pada *unit test controller* pengecekan data dilakukan melalui JSON, maka untuk *service* pengecekan dilakukan melalui `assertEquals(response,actionHistoryServiceImpl.getAllByInvoiceId(invoiceId)).verify(actionHistoryRepository,times(1)).findAllByInvoiceId(invoiceId)` berfungsi untuk memastikan, bahwa fungsi `airwayBillHistoryService.getAllByKey(key)` sudah terpanggil setidaknya satu kali dalam *unit test*.

Berikut potongan *code* pada `service upload history`:

```
@Test
public void testStoreUploadHistoryServiceSuccess() throws Exception{

    UUID uuid = UUID.fromString(uploadHistoryid);
    mockStatic(UUID.class);
    mockStatic(System.class);
    when(invoiceRepository.findOne(invoiceId)).thenReturn(invoice);
    when(UUID.randomUUID()).thenReturn(uuid);
    when(System.currentTimeMillis()).thenReturn(uploadedDateConvert);

    List<UploadHistory> uploadHistories = Arrays.asList(
            new UploadHistory(uuid.randomUUID().toString(),fileName,uploadedBy,
                    uploadedDate,status,proccesedRows, invoice: null));
    response.setData(uploadHistories.get(0));

    when(uploadHistoryRepository.save(uploadHistories.get(0))).thenReturn(uploadHistories.get(0));
    Response result = uploadHistoryService.store(invoiceId,fileName,uploadedBy,status,proccesedRows);
    assertEquals(response,result);
    verify(uploadHistoryRepository, times( wantedNumberOfInvocations: 1)).save(uploadHistories.get(0));
    verify(invoiceRepository,times( wantedNumberOfInvocations: 1)).findOne(invoiceId);
}
```

Gambar 5.43. Unit test service upload history

Gambar 5.43. merupakan *unit test* pada `service upload history` dengan menguji logika bisnis yang dilakukan pada *service*. `testStoreUploadHistoryServiceSuccess()` merupakan *unit test* untuk melakukan simpan data `upload history`. *Unit test* perlu dilakukan untuk memastikan data berhasil disimpan melalui fungsi pada *service* tersebut. Pada proses penyimpanan data, terdapat data-data yang berupa data dinamis atau dapat

berubah, seperti data waktu yang menggunakan `timestamp` dan data `id` yang menggunakan UUID.

Data dengan sifat dinamis akan di-*mock* supaya menjadi statis, sebagai contoh seperti: `mockStatic(UUID.class)` dan `mockStatic(System.class)`. Contoh potongan *code* tersebut mengembalikan objek `Response`, ketika fungsi `invoiceRepository.findOne(invoiceId)` dipanggil. Jika pada *unit test controller* pengecekan data dilakukan melalui JSON, maka untuk *service* pengecekan dilakukan melalui `assertEquals(response,result)`. Method ini, memastikan bahwa proses penyimpanan data melalui fungsi menghasilkan respon yang benar.

Berikut potongan *code* pada `service rule`:

```
@Test
public void testUpdateToleranceRuleSuccess() throws Exception{
    toleranceThirdPartyLogisticRequest.setLogisticCode(logisticCodeFirst);
    toleranceThirdPartyLogisticRequest.setDistanceTolerance(distanceToleranceFirst);
    response.setData(rules);
    response.setMessage("Add Tolerance for " + toleranceThirdPartyLogisticRequest.getLogisticCode() + " Success!");
    rule.setDistanceTolerance(0);
    when(ruleRepository.findRuleByLogisticCode(logisticCodeFirst)).thenReturn(rule);
    when(ruleRepository.findAll()).thenReturn(rules);
    Response result = ruleService.updateToleranceThirdPartyLogistic(toleranceThirdPartyLogisticRequest);
    assertEquals(response.getData(),result.getData());
    verify(ruleRepository, Mockito.times( wantedNumberOfInvocations: 1))
            .updateToleranceThirdPartyLogistic(logisticCodeFirst,distanceToleranceFirst);
    verify(ruleRepository, Mockito.times( wantedNumberOfInvocations: 1)).findRuleByLogisticCode(logisticCodeFirst);
    verify(ruleRepository, Mockito.times( wantedNumberOfInvocations: 1)).findAll();
}
```

Gambar 5.44. Unit test service rule

Gambar 5.44. merupakan *unit test* pada `service rule` dengan menguji logika bisnis yang dilakukan pada *service*. `testUpdateToleranceRuleSuccess()` merupakan *unit test* untuk melakukan mengubah data `distance tolerance` pada suatu `logistic code`. *Unit test* perlu dilakukan untuk memastikan perubahan data `distance tolerance`, berhasil dilakukan melalui fungsi pada *service* tersebut.

```
when(ruleRepository.findRuleByLogisticCode(logisticCodeFirst))
.thenReturn(rule);
when(ruleRepository.findAll()).thenReturn(rules);
```

Pada contoh potongan *code* di atas merupakan *method* yang akan mengembalikan data seperti `Rule` dan daftar `Rule` melalui `Rule Repository` yang telah di-*mock*. Jika pada *unit test controller* pengecekan data dilakukan melalui JSON, maka untuk *service* pengecekan dilakukan melalui `assertEquals(response.getData(),result.getData())`. *Method* ini,

memastikan bahwa proses pengubahan data melalui fungsi menghasilkan respon yang benar.

Berikut potongan *code* pada `service invoice`:

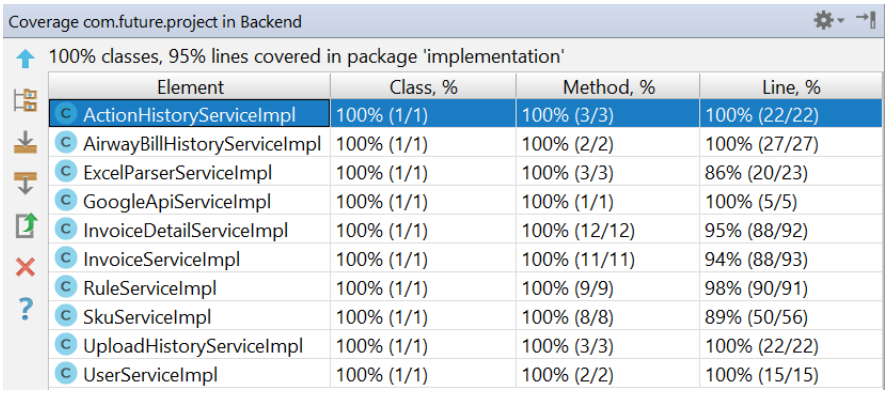
```
@Test
public void testDeleteInvoiceSuccess() throws Exception{
    when(invoiceRepository.findOne(id)).thenReturn(invoice);
    Response res = invoiceService.delete(id);
    verify(invoiceRepository, times( wantedNumberOfInvocations: 1)).delete(invoice);
    verify(invoiceRepository, times( wantedNumberOfInvocations: 1)).findOne(id);
}
```

Gambar 5.45. Unit test service invoice

Gambar 5.45. merupakan *unit test* pada `service invoice` dengan menguji logika bisnis yang dilakukan pada *service*. `testDeleteInvoiceSuccess()` merupakan *unit test* untuk melakukan hapus data `invoice` berdasarkan `id`. *Unit test* perlu dilakukan untuk memastikan hapus data `invoice` berhasil dilakukan, melalui fungsi pada *service* tersebut. `when(invoiceRepository.findOne(id)). thenReturn(invoice)`, *method* tersebut akan mengembalikan data `invoice` berdasarkan `id` melalui `invoiceRepository` yang telah di-*mock*.

```
verify(invoiceRepository, times(1)).delete(invoice);
verify(invoiceRepository, times(1)).findOne(id);
```

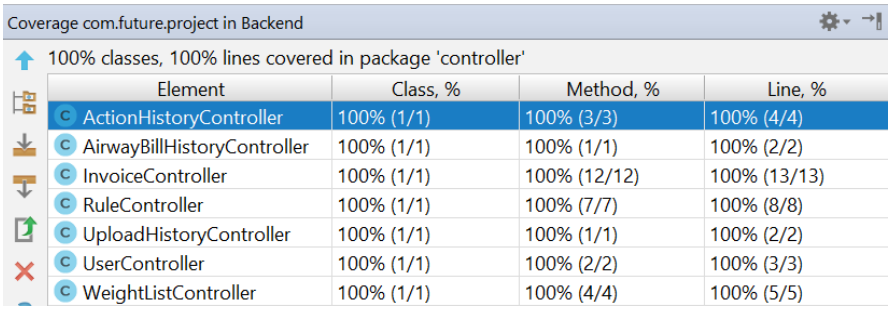
Method diatas berguna untuk memastikan bahwa fungsi yang terdapat pada *service* telah dipanggil setidaknya satu kali.



Gambar 5.46. Hasil Pengujian Package Service Implementation

Gambar 5.46. merupakan hasil pengujian berdasarkan *line coverage* dan *method coverage*, pada *package service implementation*. Hasil pengujian menunjukkan bahwa rata-rata persentase untuk *class coverage* adalah 100%. Rata-rata *method*

coverage adalah 100% dan untuk *line coverage* adalah 96,2%. Berdasarkan hasil rata-rata yang didapatkan, telah memenuhi standar pengujian perusahaan.



Element	Class, %	Method, %	Line, %
ActionHistoryController	100% (1/1)	100% (3/3)	100% (4/4)
AirwayBillHistoryController	100% (1/1)	100% (1/1)	100% (2/2)
InvoiceController	100% (1/1)	100% (12/12)	100% (13/13)
RuleController	100% (1/1)	100% (7/7)	100% (8/8)
UploadHistoryController	100% (1/1)	100% (1/1)	100% (2/2)
UserController	100% (1/1)	100% (2/2)	100% (3/3)
WeightListController	100% (1/1)	100% (4/4)	100% (5/5)

Gambar 5.47. Hasil Pengujian Package Controller

Gambar 5.47. merupakan hasil pengujian berdasarkan *line coverage* dan *method coverage*, pada *package controller*. Hasil pengujian menunjukkan bahwa rata-rata persentase untuk *class coverage* adalah 100%. Rata-rata *method coverage* adalah 100% dan untuk *line coverage* adalah 100%. Berdasarkan hasil rata-rata yang didapatkan, telah memenuhi standar pengujian perusahaan.

BAB 6

KESIMPULAN DAN SARAN

6.1. Kesimpulan

Berdasarkan hasil penelitian, pembahasan, hingga pengujian pada SIJAHA (Sistem Validasi Jarak dan Harga), maka dapat ditarik kesimpulan bahwa pembangunan SIJAHA menggunakan *Batch Processing*, berbasis web, serta fitur yang dibutuhkan pengguna berhasil dibuat. Masukan data pada proses validasi, melalui unggahan berkas(.xlsx) berhasil diterapkan. Pengambilan jarak menggunakan Google Maps API, melalui titik *longitude* dan *latitude* berhasil dilakukan, sehingga pengguna tidak perlu melakukan secara manual. Implementasi *Batch Processing* untuk memproses data, suatu *batch* melakukan eksekusi 200 data, menggunakan *framework* Spring Batch berhasil dilakukan. Jumlah rata-rata data yang mampu seorang karyawan kerjakan, per hari dengan waktu dua jam kalkulasi jarak dan harga adalah 500 data. Setelah sistem diimplementasikan, untuk 500 data, proses kalkukasi jarak dan harga membutuhkan waktu 120 detik. Melalui perbandingan terhadap proses validasi, sebelum dan sesudah sistem diimplementasikan, maka presentase untuk pengurangan waktu dalam proses validasi adalah 98.33%.

6.2. Saran

Beberapa saran untuk pengembangan Sistem Validasi Jarak dan Harga dari adalah sebagai berikut:

1. Saat ini sistem menggunakan jarak dan harga untuk melakukan perhitungan harga akhir, namun kebutuhan vendor kedepan tentu akan bertambah, sehingga perlu dibuat fitur baru seperti kelola data berat barang dan pengecekan berat barang pada data berkas(.xlsx) yang diunggah pengguna.
2. Dibuat fungsionalitas baru untuk menghasilkan berkas proses validasi dalam periode waktu tertentu.

DAFTAR PUSTAKA

Bagir, M. (2016, August). IMPLEMENTASI ONLINE TESTING DENGAN BATCH PROCESSING SYSTEM. In *Seminar Nasional Telekomunikasi dan Informatika*.

Cogoluegnes, A., Templier, T., Gregory, G., & Bazoud, O. (2011). *Spring Batch in Action*. New York: Manning Publications Co.

Copes, F. (2018). *The Vue.js Handbook*. Retrieved from https://vuehandbook.com/?ref=madewithvuejs.com

Liu, Y. (2016). Research on Business Model Innovation of Logistics Enterprises. *Modern Economy*, *7*(14), 1720.

Martin, N., Swennen, M., Depaire, B., Jans, M., Caris, A., & Vanhoof, K. (2015). *Batch processing: definition and event log identification*. RWTH Aachen University.

Minella, M.T. (2011). *Pro Spring Batch*. New York: Springer Science + Business Media.

Nanehkaran, Y. A. (2013). An introduction to electronic commerce. *International journal of scientific & technology research*, *2*(4), 190-193.

Nguyen, T. (2018). Java Spring Framework in developing the Knowledge Article Management application: A brief guide to use Spring Framework.

Shahriari, S., Shahriari, M., & Gheiji, S. (2015). E-commerce and it impacts on global trend and market. *International Journal Of Research–Granthaalayah*, *3*(4), 49-55.

Syahputra, S. (2018). Pembangunan Sistem Informasi Penghasil Berkas Laporan Menggunakan Metode Batch Processing. Thesis. UAJY.

Thiruthanigesan, K., & Thiruchchelvan, N. (2016). Data Verification and Validation Process in the Management System Development. *Middle-East Journal of Scientific Research*, *25*(5), 902-911.

Walls, C. (2016). *Spring Boot in action*. New York: Manning Publications.

Yang, X. (2014). Status of third party logistics–a comprehensive review. *Journal of logistics Management*, *3*(1), 17-20.