IMPLEMENTASI METODE SOA *(SERVICE ORIENTED ARCHITECTURE)* DALAM *PURCHASE ORDER MANAGEMENT* PADA SISTEM *E-COMMERCE* MENGGUNAKAN SPRING *FRAMEWORK* DAN JAVA EE (*ENTERPRISE EDITION*)

SKRIPSI

Diajukan Kepada:
Fakultas Sains dan Teknologi
Universitas Islam Negeri (UIN)
Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
ASRONI HIDAYAT FAHMI
NIM.11650029

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2015

IMPLEMENTASI METODE SOA *(SERVICE ORIENTED ARCHITECTURE)* DALAM *PURCHASE ORDER MANAGEMENT* PADA SISTEM *E-COMMERCE* MENGGUNAKAN SPRING *FRAMEWORK* DAN JAVA EE (*ENTERPRISE EDITION*)

SKRIPSI

Oleh :

ASRONI HIDAYAT FAHMI
NIM.11650029

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG

2015

IMPLEMENTASI METODE SOA *(SERVICE ORIENTED ARCHITECTURE)* DALAM *PURCHASE ORDER MANAGEMENT* PADA SISTEM *E-COMMERCE* MENGGUNAKAN SPRING *FRAMEWORK* DAN JAVA EE (*ENTERPRISE EDITION*)

SKRIPSI

Oleh :

ASRONI HIDAYAT FAHMI

NIM.11650029

Telah disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Linda Salma Angreani, M.T	**Fatchurrochman, M.Kom**
NIP. 19770803 200912 2 005	**NIP. 19700731 200501 1 002**

Tanggal, 8 Juni 2015

Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian, M.CS
NIP. 19740424 200901 1 008

HALAMAN PENGESAHAN

IMPLEMENTASI METODE SOA *(SERVICE ORIENTED ARCHITECTURE)* DALAM *PURCHASE ORDER MANAGEMENT* PADA SISTEM *E-COMMERCE* MENGGUNAKAN SPRING *FRAMEWORK* DAN JAVA EE (*ENTERPRISE EDITION*)

SKRIPSI

Oleh :

ASRONI HIDAYAT FAHMI

NIM. 11650029

Telah Dipertahankan di Depan Dewan Penguji Skripsi dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)

Tanggal 8 Juni 2015

Susunan Dewan Penguji		Tanda Tangan
1. Penguji Utama	: M. Ainul Yaqin, M.Kom NIP. 19761013 200604 1 004	( )
2. Ketua	: Syahiduz Zaman, M.Kom NIP. 19700502 200501 1 005	( )
3. Sekretaris	: Linda Salma Angreani, M.T NIP. 19770803 200912 2 005	( )
4. Anggota	: Fatchurrochman, M.Kom NIP. 19700731 200501 1 002	( )

Mengetahui dan Mengesahkan
Ketua Jurusan Teknik Informatika

Dr.Cahyo Crysdian, M.CS
NIP. 19740424 200901 1 008

SURAT PERNYATAAN

ORISINALITAS PENELITIAN

Saya yang bertanda tangan di bawah ini:

Nama : Asroni Hidayat Fahmi

NIM : 11650029

Fakultas/ Jurusan : Sains dan Teknologi / Teknik Informatika

Angkatan tahun/semester : 2011 / VIII

Judul : **IMPLEMENTASI METODE SOA *(SERVICE ORIENTED ARCHITECTURE)* DALAM *PURCHASE ORDER MANAGEMENT* PADA SISTEM *E-COMMERCE* MENGGUNAKAN SPRING *FRAMEWORK* DAN JAVA EE (*ENTERPRISE EDITION*)**

Menyatakan dengan sebenar-benarnya bahwa hasil penelitian saya ini tidak terdapat unsur-unsur penjiplakan karya penelitian atau karya ilmiah yang pernah dilakukan atau dibuat oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata hasil penelitian ini terbukti terdapat unsur-unsur penjiplakan, maka saya bersedia untuk mempertanggungjawabkan, serta diproses sesuai peraturan yang berlaku.

Malang, 8 Juni 2015

Yang membuat pernyataan

Asroni Hidayat Fahmi

NIM. 11650029

HALAMAN PERSEMBAHAN

Saya persembahkan karya ini kepada:
Kedua orang tuaku yang sangat luar biasa,
ayahanda Drs. Tarmidi dan ibunda Umi Thoyibah
yang telah menyayangi dan membesarkan saya

Kakak adik saya tercinta, Afri Arqomah Muninggar
dan Ilham Maulana Mulia.

Sahabat-sahabat saya Ibnu Khaldun kamar 5,
Sahabat seperjuangan saya "*super* kontrakan",
Wildan, Dani, Hafid, Galih, Alvian, Ambon, Iwan, Andi, Ucup.
Yang setia menemani setiap perjuangan dalam susah dan senang.
Teman saya yang selalu menjawab ketidaktahuan saya,
Masiti, Alvin, Juniar, Ihsan, Ulfa2, Hudan dan Semuanya...

Teman hidup saya, Miftachul Choiroh yang baik hatinya...

Tim Lab. Riset Informatika Ibu Linda Salma, Hari, Emil.
Teman-teman saya di semua jurusan terutama
di Jurusan Teknik Informatika angkatan 2011 (INTEGER '11)
yang tidak saya sebutkan satu-satu disini.
Sukses dan sehat selalu untuk semuanya...
Kepada setiap orang yang telah membantuku
dan setia menemaniku, dalam persahabatan dan cinta...

Respect, Unity, for All
Semoga bermanfaat, tetap dalam perdjoeangan!

HALAMAN MOTTO

خير الناس أنفعهم للناس

"Dan sebaik-baik manusia adalah yang paling bermanfaat bagi manusia lain" (HR. Thabrani)

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Segala puji bagi Allah SWT tuhan semesta alam, karena atas segala rahmat dan karunia-Nya kepada penulis sehingga penulis mampu menyelesaikan skripsi dengan judul "Implementasi Metode SOA *(Service Oriented Architecture)* Dalam *Purchase Order Management* Pada Sistem *E-Commerce*" dengan baik dan lancar. Shalawat dan salam selalu tercurah kepada tauladan terbaik kita Nabi Agung Muhammad SAW yang telah membimbing umatnya dari zaman kegelapan dan kebodohan menuju cahaya islam yang terang *rahmatan lil alamiin* ini.

Dalam penyelesaian skripsi ini, banyak pihak yang telah memberikan bantuan baik secara moril, nasihat dan semangat maupun materiil. Atas segala bantuan yang telah diberikan, penulis ingin menyampaikan doa dan ucapan terimakasih yang sedalam-dalamnya kepada :

1. Prof. DR. H. Mudjia Raharjo, M.Si, selaku rektor UIN Maulana Malik Ibrahim Malang beserta seluruh staf. Dharma Bakti Bapak dan Ibu sekalian terhadap Universitas Islam Negeri Malang turut membesarkan dan mencerdaskan penulis.
2. Dr. Hj. Bayyinatul M., drh., M.Si, selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang beserta seluruh staf. Bapak dan ibu sekalian sangat berjasa memupuk dan menumbuhkan semangat untuk maju kepada penulis.

3. Bapak Dr. Cahyo Crysdian, selaku Ketua Jurusan Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang, yang sudah memberi banyak memberi pengetahuan, inspirasi dan pengalaman yang berharga.
4. Ibu Linda Salma Angreani, M.T selaku dosen pembimbing I yang telah meluangkan waktu untuk membimbing, memotivasi, mengarahkan dan memberi masukan kepada penulis dalam pengerjaan skripsi ini hingga akhir.
5. Bapak H. Fatchurrochman, M.Kom, selaku dosen pembimbing II yang juga senantiasa memberi masukan dan nasihat serta petunjuk dalam penyusunan skripsi ini.
6. Ayah, Ibu, Kakak dan Adik serta keluarga besar saya tercinta yang selalu memberi dukungan yang tak terhingga serta doa yang senantiasa mengiringi setiap langkah penulis.
7. Segenap Dosen Teknik Informatika yang telah memberikan bimbingan keilmuan kepada penulis selama masa studi.
8. Teman – teman seperjuangan Teknik Informatika 2011 .
9. Para peneliti yang telah mengembangkan SOA pada sistem *e-commerce* yang menjadi acuan penulis dalam pembuatan skripsi ini. Serta semua pihak yang telah membantu yang tidak bisa disebutkan satu-satu. Terimakasih banyak.

Berbagai kekurangan dan kesalahan mungkin pembaca temukan dalam penulisan skripsi ini, untuk itu penulis menerima segala kritik dan saran yang membangun dari pembaca sekalian. Semoga apa yang menjadi kekurangan bisa

disempurnakan oleh peneliti selanjutnya dan semoga karya tulis ini bisa bermanfaat dan menginspirasi bagi kita semua. Amin.

Wassalamualaikum Wr. Wb.

Malang, 8 Juni 2015

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Fahmi, Asroni Hidayat. 2015. **Implementasi SOA (*Service Oriented Architecture*) dalam *Purchase Order Management* pada Sistem *E-Commerce* Menggunakan Spring Framework dan Java EE (*Enterprise Edition*)**. Pembimbing: (1) Linda Salma Angreani, M.T, (2) Fatchurrochman, M.Kom.

Kata Kunci : *E-Commerce, SOA, Web Service, System Integration*

E-Commerce merupakan sebuah sistem yang digunakan untuk melakukan transaksi bisnis secara online. Dalam menjalankan proses bisnisnya, sistem *e-commerce* harus berinteraksi dengan sistem yang lain. Sebagai contoh pada proses *purchase order*, dalam proses ini melibatkan sistem *billing, inventory, shipping* dan *payment*. Dibutuhkan sebuah model atau metode pengintegrasian yang bisa menunjang proses *purchase order* tersebut.

Berdasarkan permasalahan diatas peneliti menggunakan SOA untuk memecahkannya. Proses dimulai dengan membuat rancangan model SOA, implementasi *web service* dan penggabungan fitur SOA kedalam proses *purchase order* pada sistem *e-commerce*. Dalam pengembangan sistem *e-commerce* ini menggunakan pendekatan *System Development Life Cycle* (SDLC) dengan beberapa tahapan yaitu analisis dan perancangan, implementasi, pengujian serta pemeliharaan. Perancangan dimodelkan dengan UML (*Unified Modeling Language*). Bahasa yang digunakan dalam penelitian ini adalah *Java*, dengan *platform* Java EE (*Enterprise Edition*) dengan *framework* Spring, Hibernate ORM dan Metro Web Service.

Hasil dari penelitian ini adalah sistem *e-commerce* terintegrasi yang mendukung otomatisasi pada proses *purchase order*. Sistem diintegrasikan untuk mengatasi masalah *loosely coupled* dan *interoperability* sistem.

ABSTRACT

Fahmi, Asroni Hidayat. 2015. The **Implementation of SOA (Service Oriented Architecture) for Purchase Order Management in E-Commerce System Using Spring Framework and Java EE (Enterprise Edition)**. Supervisor: (1) Linda Salma Angreani, M.T, (2) Fatchurrochman, M.Kom.

Keywords : E-Commerce, SOA, Web Service, System Integration

E-Commerce is a system used to conduct business transactions online. In carrying out its business processes, e-commerce system should interact with other systems. For example, the purchase order process, in this process involves billing systems, inventory, shipping and payment. It takes a model or method that can support the integration process of the purchase order.

Based on the above problems researchers using SOA to solve it. The process begins by making the SOA model design, implementation web service and SOA features incorporation into the purchase order to the e-commerce system. In the development of e-commerce system is using the System Development Life Cycle (SDLC) with several stages of analysis and design, implementation, testing and maintenance. The design is modeled with UML (Unified Modeling Language). The language used in this study is Java, the platform Java EE (Enterprise Edition) with the Spring framework, Hibernate ORM and Metro Web Service..

Results of this research are integrated e-commerce system that supports the automation of the purchase order process. Integrated system to address the problems loosely coupled and interoperable systems.

BAB I

PENDAHULUAN

1.1. Latar Belakang

Secara istilah jual beli/bai' berarti: saling tukar-menukar harta dengan tujuan kepemilikan (Tarmizi, 2012). Allah menghalalkan jual beli dan mengharamkan riba. Hal ini sesuai dengan firman Allah SWT dalam surat Al-Baqarah ayat 275:

وَأَحَلَّ اللَّهُ الْبَيْعَ وَحَرَّمَ الرِّبَا

"*Allah telah menghalalkan jual beli dan mengharamkan riba*" (Al-Baqarah: 275)

Berdagang adalah profesi yang mulia dalam Islam. Buktinya Rasulullah SAW sendiri adalah pedagang dan beliau memuji serta mendoakan para pedagang yang jujur. Dari Abu Sa'id Al-Khudri radhiyallahu 'anhu, Nabi shallallahu alaihi wasallam bersabda:

التاجر الصدوق الأمين مع النبيين والصديقين والشهداء

"*Pedagang yang senantiasa jujur lagi amanah akan bersama para nabi, orang-orang yang selalu jujur dan orang-orang yang mati syahid*." (HR. Tirmidzi).

Salah satu bentuk berdagang di era internet sekarang ini adalah *e-commerce*. Dengan *e-commerce* dapat dilakukan transaksi jual beli barang melalui internet.

E-Commerce begitu berkembang pesat di Indonesia, hal ini dapat dilihat dari pendapatan transaksi *E-Commerce* di Indonesia yang mencapai USD 120 juta pada 2010 dan diprediksi akan menjadi USD 650 juta pada 2015 ("Indonesia Telecom Outlook – Go Online", 2012). Dalam transaksi *E-Commerce* terdiri dari beberapa proses diantaranya *Purchase Order Management* (POM), *Inventory Management* (IM) dan *Customer Relationship Management* (CRM) dan lain-lain (Ezenwoke dkk, 2013). Salah satu proses yang menjadi fokus peneliti dalam penelitian ini adalah POM. POM adalah proses dimana *customer* melakukan pemilihan produk terhadap barang yang ingin dia beli (*Purchasing*), proses pemesanan barang dan pembayaran serta pengiriman (*Ordering*).

Di Sistem *e-commerce* konvensional, proses inti dari POM ini terpusat ke dalam satu aplikasi, misalnya sistem *inventory* barang yang menjadi satu dengan *web portal* dan tidak terhubung dengan sistem yang lain, misalnya ke sistem perusahaan pengiriman barang, sistem pembayaran bank dan lain-lain. Akibatnya, semua data menjadi terpusat, pemrosesan transaksi pelanggan menjadi lama karena sistem tidak terotomatisasi dan terintegrasi antara satu dengan yang lainnya. Hal ini tentunya juga berakibat kepada kepuasaan belanja pelanggan, serta pendapatan perusahaan *e-commerce* itu sendiri.

Kemudian dirancanglah *e-commerce* terintegrasi. Dalam *e-commerce* jenis ini, proses bisnis POM tersebar ke bagian-bagian atau departemen-departemen yang berbeda-beda. Diantaranya, bagian *shipping* (pengiriman), bagian *inventory* (gudang), bagian *billing* (penagihan) (Baraka, Al Asqar, 2013). Setiap departemen memiliki tugasnya masing-masing dengan aplikasi tersendiri yang dibangun

dengan database, sistem operasi dan bahasa pemrograman yang berbeda-beda. Permasalahan dan tantangan yang muncul adalah integrasi proses bisnis dari setiap bagian atau departemen yang ada. Dibutuhkan metode untuk mengintegrasikan proses tersebut. Penelitian sebelumnya yang dilakukan oleh Baraka, Al-Aqshar (2013) telah merancang model integrasi dan implementasi SOA dalam POM. Akan tetapi dalam model tersebut belum bisa memproses beberapa jenis barang yang berbeda kedalam satu *order*.

Maka berdasarkan masalah diatas, peneliti mengusulkan untuk membuat sebuah "Implementasi Metode SOA(*Service Oriented Architecture)* dalam *Purchase Order Management* Pada Sistem *E-Commerce*" untuk mengintegrasikan, serta mengotomatisasikan proses transaksi POM, terutama dalam proses *purchase order* dengan berbagai jenis barang.

1.2. Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah pada penelitian ini yaitu :

1. Apakah SOA dapat diterapkan dalam Sistem *E-Commerce*?
2. Bagaimana merancang Sistem *E-commerce* menggunakan SOA dalam *Purchase Order Management* dengan pendekatan berorientasi objek?
3. Bagaimana implementasi SOA dalam *Purchase Order Management* pada Sistem *E-Commerce*?

1.3. Tujuan

Adapun tujuan dari penelitian ini, yaitu :

1. Membuktikan bahwa metode SOA dapat digunakan dalam Sistem *E-Commerce.*
2. Merancang sistem *E-Commerce* menggunakan SOA dengan pendekatan berorientasi objek.
3. Mengimplementasikan Sistem *E-Commerce* dengan metode SOA.

1.4. Batasan Masalah

Batasan masalah dalam penelitian ini meliputi :

1. Tidak membahas tentang keamanan Sistem SOA, penelitian ini dibatasi pada implemetasi SOA.
2. Berfokus pada tranksaksi *Business-to-consumer* (B2C).
3. Berfokus pada proses *Purchase Order Management* (POM).
4. Pembayaran hanya menggunakan kartu kredit.

1.5. Manfaat

Adapun manfaat dari penelitian ini antara lain :

1. Menghasilkan Sistem *E-Commerce* terintegasi menggunakan SOA.
2. Mempercepat proses tansaksi *Purchase Order Management* (POM) dengan otomatisasi dan integrasi proses bisnis.
3. Meningkatkan mutu pelayanan perusahaan *E-Commerce.*
4. Meningkatkan kepuasan belanja pelanggan.

BAB II

TINJAUAN PUSTAKA

2.1. Tinjauan Pustaka

Penelitian sebelumnya yang berkaitan dengan pengembangan sistem menggunakan SOA telah banyak dilakukan.

Penelitian pertama dilakukan oleh Arba dan Buchman (2009) dengan judul "*Using SOA for E-Commerce Systems Development*". Dalam penelitian ini dibangun model *E-Commerce* dengan menggunakan SOA akan tetapi masih secara umum tidak spesifik pada fitur tertentu. Kelebihan dalam model ini menunjukkan bahwa sistem *E-Commerce* yang dibangun menggunakan metode SOA dapat mengekspos fungsionalitas bisnis serta sangat adaptif terhadap perubahan teknologi. Penggunaan metode SOA membuat sistem berjalan flexible dan memiliki daya interoperabilitas yang tinggi.

Penelitian kedua yang dilakukan oleh Safuwan dkk (2010) yang berjudul "Integrasi Perangkat Lunak *Enterprise Resource Planning* (ERP) Dengan Menggunakan Metode *Service Oriented Architecture* (SOA)". Dalam penelitian ini dibangun sistem integrasi ERP dengan menggunakan SOA tujuannya untuk memudahkan kolaborasi data dengan berbagai ERP dan pengimplementasian teknis menggunakan OpenESB dan BPEL. Integrasi antara *legacy system* dapat dilakukan dengan mudah jika sudah menggunakan arsitektur SOA. Penggunaan ESB merupakan teknologi yang sangat mendukung implementasi sistem terintegrasi SOA(Safuwan dkk, 2010).

Penelitian ketiga yang dilakukan oleh Sarwosri dan Farah Naja (2011) yang berjudul "Rancang Bangun Perangkat Lunak Aplikasi Pelayanan Kesehatan Berbasis *Service-Oriented Architecture*". Dalam aplikasi pelayanan kesehatan ini, implementasi integrasi *service* dengan SOA dapat memudahkan pasien untuk menelusuri rekam medisnya dari semua rumah sakit yang pernah dia kunjungi, tanpa perlu mengunjungi satu per satu rumah sakit tersebut. Pasien juga akan lebih mudah mendapatkan info dan fasilitas kesehatan yang dibutuhkan tanpa perlu mengunjungi satu per satu web rumah sakit yang ada. Sistem ini dibangun diatas *platform* .NET, dengan bahasa C# dan database Oracle XE. Dalam pengimplementasian SOA ini, peneliti menyarankan untuk meningkatkan keamanan data dari *service* yang ada. Sehingga datanya tidak bisa diakses oleh pihak lain yang tidak berwenang, mengingat akan sensitifnya data yang ada.

Penelitian keempat yang dilakukan oleh Azizi Khoirul Haq (2012) yang berjudul "Rancang Bangun Sistem Informasi Apotek Terintegrasi Menggunakan *Service Oriented Architecture*". Dalam penelitian ini dibangun Sistem Informasi Apotek Terintegrasi dimana member dapat melakukan pemesanan obat di apotek tertentu dan apotek sekitarnya dengan melakukan *direct order*. Sistem dibangun dengan *framework Codeigniter* dan dengan *library* NuSOAP. Jenis integrasi *service* dalam penelitian ini adalah *point to point*, karena tidak menggunakan *Enterprise Service Bus*(ESB) dalam melakukan *routing service* yang ada. Sistem ini tersedia dalam versi *Web, Desktop* dan *Mobile* atau *J2ME*.

Dari berbagai studi literature di atas, SOA dapat mengatasi masalah integrasi *service* dengan sangat baik. Dalam penelitian ini, peneliti akan menggunakan konsep integrasi SOA dengan *model point-to-point*.

2.2. Landasan Teori

2.2.1 . *Service Oriented Architecture (SOA)*

Menurut Erl, SOA merupakan arsitektur perangkat lunak yang dibangun menggunakan prinsip-prinsip perancangan berorientasi *service*, sedangkan orientasi *service* merupakan konsep dalam rekayasa perangkat lunak yang merepresentasikan pendekatan berbeda untuk memisahkan kepentingan (Erl, 2005). Hal ini berarti bahwa fungsionalitas sistem dipecah ke dalam unit logic yang lebih kecil yang dinamakan dengan *service*. *Service-service* ini lepas satu sama lain, tetapi mempunyai kemampuan untuk berinteraksi satu sama lain melalui mekanisme komunikasi tertentu. Karena itu, Erl (2005) mendefinisikan komponen SOA sebagai *service*, *description*, dan *messages*. *Service* berkomunikasi dengan yang lain melalui *message* yang memungkinkan interaksi antara *services*, yang ditetapkan oleh *description*. Dua *service* berkomunikasi satu sama lain yang diacu sebagai *service requestor* dan *service provider*. *Service requestor* adalah *service* yang memanggil *service* lain, sedangkan yang dipanggil disebut *service provider*.

Service sendiri dapat dipandang sebagai enkapsulasi lojik dari satu atau sekumpulan aktivitas tertentu. Otomatisasi proses bisnis merupakan sekumpulan aktivitas yang disusun dalam langkah-langkah sebagai implementasi proses bisnis. Setelah seluruh permasalahan dapat dibagi dalam beberapa *service*, solusi dari

permasalahan tersebut harus bisa diselesaikan dengan memungkinkan seluruh *service* berpartisipasi dalam sebuah orkestrasi. Untuk itu ada beberapa permasalahan yang harus dimiliki oleh *service* yaitu bagaimana *service* saling berhubungan, bagaimana *service* berkomunikasi, bagaimana *service* didesain, dan bagaimana pesan antar *service* didefinisikan (Erl, 2005).

Pembagian berdasarkan *service* ini sesungguhnya bukan sesuatu yang baru, karena telah banyak diterapkan. Namun hal baru dari pendekatan *service oriented* ini terkait sifat-sifat yang dimilikinya (Erl, 2005), yaitu :

1. *Loosly coupled*, yaitu setiap *service* berdiri sendiri secara independen dan tidak bergantung dengan *service* lain untuk berjalan. Ketergantungan diminimalisir sehingga hanya butuh mekanisme komunikasi satu sama lain.
2. *Service contract*, yaitu setiap *service* memiliki kesepakatan mengenai cara untuk berkomunikasi.
3. *Autonomy*, yaitu *service* memiliki hak penuh terhadap semua lojik yang dienkapsulasinya.
4. *Abstraction*, yaitu *service* tidak memperlihatkan bagaimana lojik yang diimplementasikan di dalamnya.
5. *Reusability*, yaitu lojik dibagi menjadi sekumpulan *service* yang dapat digunakan lagi tanpa membuat ulang.
6. *Statelessness*, yaitu *service* tidak memiliki status tertentu terkait dengan aktivitas yang dilakukannya.
7. *Discoverability*, yaitu *service* didesain secara deskriptif sehingga bisa ditemukan dan diakses melalui mekanisme pencarian tertentu.

SOA terdiri atas sekumpulan *service*. Namun sekumpulan *service* tidak cukup untuk membentuk sebuah arsitektur ini. Menurut (Erl, 2005), SOA terdiri atas empat komponen, yaitu:

1. *Message*, yaitu data yang dibutuhkan untuk menyelesaikan sebagian atau sebuah unit kerja, yang dipertukarkan antara satu *service* dengan *service* yang lain.
2. *Operation*, yaitu fungsi-fungsi yang dimiliki oleh sebuah *service* untuk memproses *message* hingga menghasilkan sesuatu. Fungsi-fungsi inilah yang nantinya akan saling berinteraksi untuk menyelesaikan sebuah unit kerja.
3. *Service*, mempresentasikan sekumpulan *operation* yang berhubungan untuk menyelesaikan sekumpulan unit kerja yang berhubungan.
4. *Process*, merupakan *business rule* yang menentukan operasi mana yang digunakan untuk mencapai tujuan tertentu.

2.2.2. *E-Commerce*

Menurut (Loudon, 2012) *e-commerce* adalah penggunaan internet sebagai media untuk transaksi bisnis. Lebih luasnya, *e-commerce* adalah transaksi bisnis secara digital antara organisasi atau individu. Transaksi ini berjalan di jaringan internet dan melalui *website*, dengan melakukan transaksi komersial seperti pertukaran uang untuk membeli barang atau jasa. *E-commerce* tumbuh begitu cepat. Hal itu disebabkan teknologi internet dan *website* yang digunakan memiliki pengaruh yang lebih hebat dibandingkan teknologi jaman dahulu seperti radio, televisi dan lain-lain. E-commerce mampu mentransformasi proses didunia nyata

yang masih manual dan tersekat-sekat menjadi *platform* bisnis berskala global yang bisa diakses oleh siapapun, kapanpun dan dimanapun. Sebagai contoh, sebuah tempat pusat perdagangan seperti mall, toko retail barang yang kemudian berubah menggunakan sistem *e-commerce marketplace*.

2.2.2.1. Keuntungan Menggunakan *E-Commerce*

Beberapa keuntungan dari penggunaan sistem *e-commerce* :

1. *Global Search*

   Dengan teknologi *e-commerce* transaksi bisnis menjadi lebih luas jangkauannya, lintas negara lintas budaya. *E-commerce* menjadikan biaya transaksi lebih murah dibandingkan dengan jual beli secara tradisional.

2. *Universal Standard*

   E-commerce berjalan diatas jaringan internet yang sudah terstandarisasi. Baik dalam hal protokol komunikasi, perangkat komputer dan jaringan, perangkat lunak dan lain-lain termasuk juga keamanan dalam transaksi menjadi standar yang sangat diperhatikan dalam sistem *e-commerce* ini.

3. *Reachness*

   Kekayaan informasi yang disediakan situs *ecommerce* membuat orang lebih mudah mengakses informasi tersebut secara cepat dan jelas.

4. *Interactivity*

   Teknologi *e-commerce* menyediakan alur komunikasi dua arah antara pedangan dengan pembeli. Semua proses berjalan secara terbuka, sehingga dapat memperlancar proses transaksi.

5. *Information Destinity*

   Karena *e-commerce* menggunakan *platform* internet. Maka informasi akan lebih mudah diakses dan sampai kepada pelanggan.

6. *Personalization / Customization*

   Teknologi *e-commerce* memiliki *personalization,* maksudnya pedagang dapat mengirim pesan pemasaran ke individu tertentu berdasarkan nama, ketertarikan dan pesananan terakhir. *Customization*, artinya pelanggan dapat merubah pilihan barang atau jasa sesuai yang dikehendaki yang terdapat pada pengaturan *preference* pada sistem *e-commerce*.

7. *Social Technology*

   Dalam pemasaran *e-commerce* , *social media* bertindak sebagai penguat *marketing*. Besarnya komunitas online di dunia maya serta akses internet yang memadai membuat marketing menjadi lebih mudah dan efisien.

2.2.2.1. Klasifikasi *E-Commerce*

Klasifikasi *e-commerce* menurut jenis pelakunya antara lain sebagai berikut:

1. *Business-to-consumer (B2C)*

   E-Commerce yang melibatkan penjualan produk atau layanan secara eceran kepada pembeli individu. Misalnya situs blibli.com yang menjual smartphone, tas ke konsumen individu.

2. *Business-to-business (B2B)*

   E-Commerce yang melibatkan penjualan barang atau jasa antara partner bisnis. Sebagai contoh situs alibaba yang merupakan *B2B marketplace*.

3. *Consumer-to-consumer (C2C)*

   E-Commerce yang melibatkan konsumen yang menjual lagi ke konsumen yang lain. Sebagai contoh situs ebay.com dimana konsumen dapat menjual barang mereka kepada konsumen lain dengan penawaran yang lebih tinggi atau harga yang sesuai dengan yang dikehendaki.

2.2.3. *Web Service*

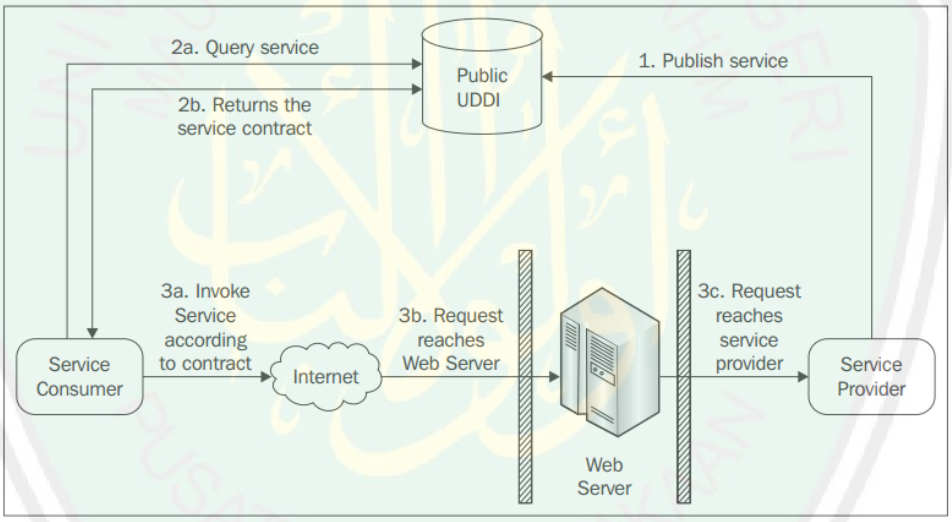
Web services adalah sebuah teknologi yang mengijinkan untuk membuat aplikasi yang independen terhadap *platform* pengembangan. *Web services* dapat dikembangkan dengan beragam bahasa dan beragam *platform* yang mengikuti standar teknologi. Sebuah *Web services* merupakan suatu aplikasi yang membuka kode fungsionalitas aplikasi ke beberapa aplikasi (Arora dan Kishore, 2002).

Web service adalah sepotong logika bisnis, yang dapat diakses melalui internet dengan standar protocol seperti HTTP atau SMTP. *Web service* merupakan sebuah teknologi yang dapat digunakan untuk mengimplementasikan layanan (Salter-Jennings, 2008). Menurut Hwang et al (2008), WS telah menjadi standard untuk melakukan ekspose fungsi dari aplikasi bisnis, WS menjadi kerangka untuk pengembangan aplikasi menggunakan SOA. WS secara efektif mampu mengintegrasikan beberapa WS kedalam sebuah komposit atau aplikasi ESB.

WS telah banyak digunakan untuk membangun aplikasi berbasis SOA. Hal ini yang menyebabkan WS identik dengan SOA. Atas dasar ini Erl (2005) menyebutkan sebagai *Contemporary* SOA. Contemporary SOA merupakan SOA

yang menggunakan WS dan XML dalam implementasinya. Untuk selanjutnya, yang disebut dengan SOA adalah *Contemporary* SOA.

Menurut W3C *Web service* merupakan aplikasi perangkat lunak yang diidentifikasi melalui URI yang antarmuka dan binding nya mampu diidentifikasi, dideksripsikan dan ditemukan melalui XML dan mendukung interaksi langsung dengan perangkat lunak lain menggunakan pesan berbasis XML melalui protocol berbasis internet atau HTTP.



Gambar 2.1. Arsitektur *Web Service* (Juric, 2007)

Komponen *Web Service* terdiri dari tiga yaitu SOAP, WSDL, dan UDDI :

1. SOAP (*Simple Object Access Protocol*)

   Dalam pertukaran pesan dalam WS diperlukan suatu standard format pesan antara peminta layanan (*requestor*) dengan penyedia layananan (*provider*). SOAP adalah format pesan yang digunakan untuk komunikasi tersebut. SOAP mendefinisikan format pembungkusan pesan dalam WS. Dalam SOAP dengan XML mengandung tiga elemen yakni *Envelope, Header* dan *Body*.

Elemen *Envelope* sebagai surat yang ditulis dalam dokumen XML yang berisi *Header* (*optional*) dan *Body* (*required*) sebagai *SOAP Message*. Elemen Header merupakan fitur tambahan pada SOAP, biasanya pada elemen header ini disematkan sebuah kunci/sandi atau *token*. Elemen body berisikan deskripsi pesan yang akan dikirim dalam bentuk *request message* dan akan dikembalikan dalam bentuk *response message*. Komunikasi ini berjalan dalam protocol HTTP pada jaringan computer.

2. *WSDL (Web Services Description Language*)

WSDL adalah sebuah format XML untuk mendeskripsikan *service* yang berada pada jaringan computer sebagai sebuah *endpoint* yang di dalamnya terdapat dokumen, fungsi atau prosedur sebuah proses aplikasi. Dengan mengacu pada dokumen WSDL aplikasi lain akan bisa mengetahui operasi-operasi apa saja yang bisa dijalankan dan data apa saja yang bisa didapatkan.

Dalam definisi tersebut termasuk juga desksripsi layanan dan fungsi-fungsi yang disediakan oleh WS.

a. *Types*

Elemen ini mendefinisikan tipe data yang terdapat dalam pesan yang dipertukarkan sebagai bagian dari layanan. Type yang diacu dalam elemen dokumen pesan WSDL didefinisikan dalam elemen type dokumen WSDL.

b. *Message*

Elemen ini mendefinisikan pesan atau data dari layanan yang akan dikomunikasikan. Dokumen WSDL mempunyai elemen pesan yang tiap pesannya dipertukarkan.

c. *Port Type*

Elemen ini menetapkan secara abstrak sekumpulan operasi yang merupakan bagian dari *service*. Dalam dokumen WSDL mempunyai satu atau lebih definisi *Port Type*.

d. *Binding*

Elemen ini menjelaskan spesifikasi protokol dan format pesan untuk jenis *port type* tertentu

e. *Port*

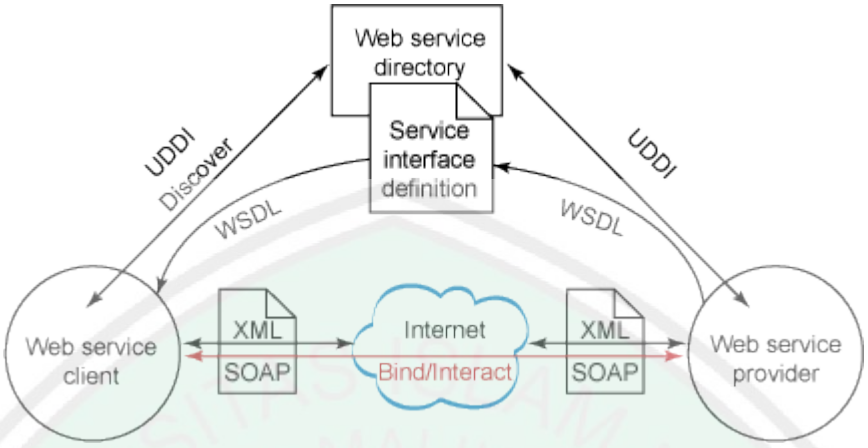
Elemen ini menjelaskan sebuah *endpoint* didefinisikan sebagai sebuah kombinasi dari sebuah binding dan alamat jaringan

f. *Service*

Elemen ini merupakan koleksi dari serangkaian *endpoint*.

3. *UDDI Registry (Universal Description, Discovery and Integration)*

Menurut W3C UDDI adalah kerangka platform independen yang berfungsi untuk mendeskripsikan, menemukan serta mengintegrasikan *service* bisnis. UDDI adalah spesifikasi standar untuk pendaftaran suatu WS. UDDI mendefinisikan metadata dan protocol yang digunakan untuk mengetahui dan mengubah informasi dari suatu WS. Langkah pertama dalam menemukan WS adalah dengan meminta alamat tempat penyimpanan dari WS yang akan dipakai yang biasa disebut dengan direktori. Setelah menemukan direktori, peminta WS dapat mengirimkan permintaan lagi untuk mendapatkan informasi detail tentang WS (misal penyedia WS dan dimana diletakkan). Selanjutnya perangkat lunak menggunakan informasi yang didapat untuk secara dinamis mengakses WS yang diinginkan.



Gambar 2.2. Arsitektur UDDI

Tempat untuk menyimpan UDDI dibagi kedalam tiga cara, yakni:

1. *Public UDDI*, dianggap sebagai *resource* bagi WS berbasis internet. Salah satu contohnya adalah *UDDI Business Registry* (UBR) yang dimiliki oleh grup seperti IBM, Microsoft, SAP
2. *Intra Entreprise UDDI*, merupakan tempat penyimpanan yang private yang menyediakan layanan bagi kalangan internal suatu perusahaan.
3. *Inter Enterprise UDDI*, merupakan repository yang dapat diakses oleh perusahaan yang merupakan partner dalam suatu bisnis.

Seperti yang telah dijelaskan terdahulu, penemuan layananan memainkan peranan yang penting dalam SOA. Banyak cara yang dapat dipakai untuk mekanisme ini, tapi dalam teknologi WS, UDDI menyediakan standard yang baik dan fleksibel untuk penemuan WS.

2.2.4. Spring Framework

Spring Framework merupakan salah satu solusi untuk mengembangkan aplikasi *enterprise* menggunakan Java. Dalam lingkup java, Spring mampu menangani infrastruktur teknologi yang berbeda-beda, sehingga kita dapat fokus pada pengembangan aplikasi. Spring memungkinkan anda untuk membangun aplikasi dengan POJOs. Kemampuan ini berlaku untuk model pemrograman Java SE dan secara penuh pada Java EE. Spring menggunakan teknik *depedency injection* (DI) dan *invention of control* (IoC), dengan ini semua *library* berbasis java dapat dikolaborasikan, dengan hasil koding yang lebih rapi dan tersturktur. Beberapa keuntungan menggunakan spring framework, antara lain :

1. Transaction Management : Spring framework menyediakan sebuah layer abstrak yang generik untuk manajemen transaksi.
2. JDBC Exception Handling : layer abstrak JDBC menawarkan exception yang bersifat hierarki sehingga memudahkan penanganan error.
3. Integration with ORM (Hibernate, JPA) : Spring menawarkan layanan integrasi pengaksesan database terbaik dengan *hibernate* dan *library persistence* lainnya.
4. AOP Framework : Spring merupakan framework yang membagi *class* berdasarkan aspek-aspek tertentu.
5. MVC Framework : Spring MVC dibangun diatas inti spring. Spring MVC merupakan framework yang sangat fleksibel dalam pengaturan layout tampilan, dan mengakomodasi beberapa teknologi view seperti JSP dan Apache Tiles.

2.2.4.1. *Dependency Injection* dan *Invention of Control (IoC)*

Aplikasi yang dibangun di java pada dasarnya memiliki beberapa layer, setiap layer melibatkan komponen yang berbeda-beda dan bergabung menjadi satu untuk membentuk aplikasi yang tepat. Dengan demikian objek aplikasi memiliki ketergantungan satu dengan yang lain. Spring menyatukan beberapa komponen untuk menjadi satu kesatuan yang koheren. Dengan penggunaan DI di Spring aplikasi java bisa diinjek dengan anotasi tertentu tergantung dengan jenis spring yang digunakan, komponen *Spring Framework IoC* menyediakan kerangka untuk menyusun komponen yang berbeda-beda tersebut kedalam sebuah aplikasi penuh yang siap untuk digunakan. Banyak perusahaan yang menggunakan Spring ini untuk membangun aplikasi *enterprise* dengan java karena kuat dan *maintainable*.

2.2.4.2. Project Spring

Spring menyediakan setiap komponen secara terpisah. Setiap komponen di Spring bisa dikolaborasikan bisa juga digunakan beberapa saja, semua tergantung kebutuhan pengembang. Komponen di Spring tertuang dalam *Spring Project (http://spring.io/projects)*, berikut beberapa diantaranya:

1. *Spring Framework*

   Merupakan inti dari komponen Spring, menyediakan dukungan penuh untuk *dependency injection*, *transaction management*, *web apps*, *data access*, *messaging*, *service* dan lain lain. Didalam Spring Framework ini terdapat komponen Spring MVC yang digunakan untuk membuat aplikasi *web enterprise* dengan java yang menggunakan konsep MVC (*Model View Controller*)

2. *Spring Data*

   Menyediakan sebuah metode pengaksesan database, baik dalam bentuk *relasional*, *non relasional*, *map-reduce* dan sejenisnya.

3. *Spring Integration*

   Merupakan implementasi *Enterprise Application Integration* di Spring. Fungsinya adalah sebagai pengaturan pertukaran pesan antara aplikasi *enterprise*.

4. *Spring Security*

   Merupakan *framework* yang mendukung authentifikasi dan otorisasi. Framework ini khusus digunakan untuk mengamankan aplikasi java berbasis spring yang telah dibuat.

5. *Spring Web Flow*

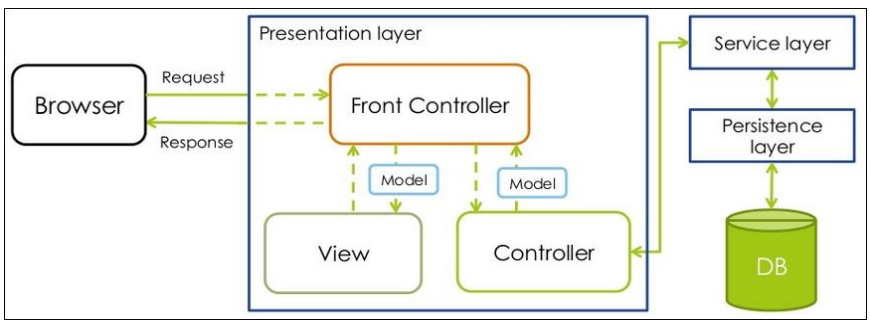
   Merupakan *framework* yang digunakan untuk mengatur alur aplikasi web, sebagai contoh alur *checkout* pada aplikasi *e-commerce*.

6. *Spring Web Service*

   Memfasilitasi pembuatan *web service* berbasis SOAP.

2.2.4.3. MVC (*Model View Controller*)

Pola desain MVC (*Model-View-Controller*) memberikan pemecahan permasalahan *coupling* yang tinggi tersebut dengan men-*decoupling* lapisan *data access*, *business logic*, dan *data presentation*. Skema MVC lebih diperjelas lagi dalam spring, sebagaimana dijelaskan pada gambar berikut ini.



Gambar 2.3. Spring MVC Architecture (Amutan, 2014)

1. *Model*

Model merepresentasikan struktur data dalam database. Spring dengan pendekatan ORM, tabel dirubah menjadi sebuah *class*. Satu tabel mewakili satu class kemudian nama kolom dalam tabel akan diubah menjadi attribute dalam class, jenis tipe data pada tabel database akan dirubah ke jenis tipe data yang ada dalam class java. Sehingga jika pengembang membangun aplikasi dengan pendekatan ini maka untuk mengakses tabel hanya perlu mengakses *class*. Pengembang akan dihadapkan dengan *object-object* java bukan *database* secara langsung.

2. *View*

View merupakan bagian yang berfungsi menampilkan data dari model. View bertugas memberikan tampilan data kepada user. View dalam aplikasi java web dibangun dengan JSP (*Java Server Page*) dan Apache Tiles sebagai *tamplate engine* nya. Dalam arsitektur spring MVC diatas, view berada dalam blog presentation layer.

3. *Controller*

Controller berfungsi menerima request dan response, setiap request yang diterima akan diproses dengan model yang dikehendaki. Setelah itu model akan mengirim data dan controller mengolah data tersebut atau langsung mengirimkan data tersebut ke pada view.

Arsitektur *Model-View-Controller* adalah sebuah pola perancangan dan pembangunan aplikasi yang memudahkan dan terstuktur. Sehingga program lebih

mudah dibaca dan dikembangkan, arsitektur MVC telah menjadi standard yang dipakai dalam pembuatan aplikasi berbasis web.

BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1. Analisis Kebutuhan Sistem

Pada fase analisis kebutuhan sistem ini dilakukan analisa terkait proses bisnis yang terjadi pada sistem *e-commerce*. Analisa pada fase ini berkaitan dengan fungsionalitas sistem yang nantinya akan digambarkan dengan diagram UML (*Unified Modeling Language*).

Penjelasan fungsionalitas sistem *e-commerce* berfokus pada transaksi *purchase order management* yang bisa dibagi menjadi dua bagian, yakni *purchasing* dan *ordering*. Berikut daftar proses bisnis POM yang ada dalam *e-commerce* (Ezenwoke, 2013) :

1. Pembelian (*Purchasing*)
    a. Melihat barang di katalog (*browse catalog*)
    b. Memasukkan barang ke keranjang (*add to cart*)
    c. Proses *checkout*
2. Pemesanan (*Ordering*)
    a. Membuat order baru (*create new order*)
    b. Informasi pengiriman barang (*shipping information*)
    c. Proses pembayaran (*payment process*)
    d. Notifikasi pemesanan (*order notification*)

Selain proses *purchase order* diatas, peneliti menambahkan beberapa proses yang diadopsi pada situs *e-commerce* B2C terkemuka di Indonesia. Kemudian dilakukan re-*engineering* proses dari situs tersebut. Proses tersebut terbagi menjadi dua yakni *customer* dan *administrator*. Didalam dua bagian tersebut terdapat proses bisnis yang nantinya akan diimplementasi dalam bentuk modul sistem. Berikut proses-prosesnya :

1. Pelanggan (*customer*)
    1. Registrasi dan *login*
    2. Merubah data akun (*change account*)
    3. Melihat daftar pemesanan (*order history*)
2. Administrator
    a. *Login*
    b. Manajemen barang (*product manager*)
    c. Manajemen pelanggan (*customer manager*)
    d. Manajemen penagihan (*billing manager*)
    e. Manajemen pemesanan (*order manager*)

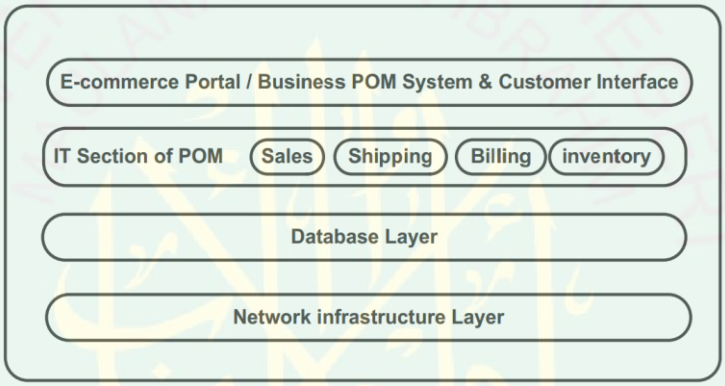
Setelah analisis kebutuhan fungsional terdefinisi berikutnya analisis non fungsional. Analisis non fungsional meliputi pihak-pihak yang berinteraksi terhadap sistem yang berjalan yaitu pengguna sistem tersebut. Terdapat 3 pengguna yaitu pengunjung, *customer* dan *administrator*.

3.2. Perancangan Sistem

Pada tahap ini dilakukan perancangan sistem berdasarkan analisis kebutuhan yang ada sebelumnya.

3.2.1. Model Teknis POM

Proses POM pada sistem *e-commerce* melibatkan berbagai departemen. Setiap departemen berada dalam tempat dan sistem yang berbeda-beda akan tetapi mereka saling terhubung untuk mendukung proses POM dalam sistem *e-commerce*. Secara fisik masing-masing departemen memiliki susunan layer sebagaimana gambar 3.1.



Gambar 3.1. Model Teknis POM (Baraka & Al-Ashqar, 2013)

1. *Portal Interface Layer*

Pada layer ini merepresentasikan sebuah antarmuka yang akan digunakan oleh pelanggan dalam melakukan transaksi POM. Antarmuka ini disajikan dalam bentuk tampilan web yang bisa diakses oleh pelanggan melalui *web browser*. Dalam antarmuka ini terdapat beberapa fitur yang mewakili fungsi-fungsi dari POM.

2. *IT Section of POM*

Pada layer ini menggambarkan bagian atau departemen dari proses POM yang meliputi : penjualan (*sales*), pengiriman (*shipping*), penagihan (*billing*) dan gudang (*inventory*). Setiap proses diwakili oleh satu departemen atau bagian dengan aplikasinya masing-masing yang dibangun menggunakan bahasa pemrograman, database dan sistem operasi yang berbeda-beda yang saling bertukar data dan berinteraksi satu sama lain. Agar POM dapat berjalan, masing-masing departemen dalam layer ini haruslah terhubung dengan SOA.

3. *Database Layer*

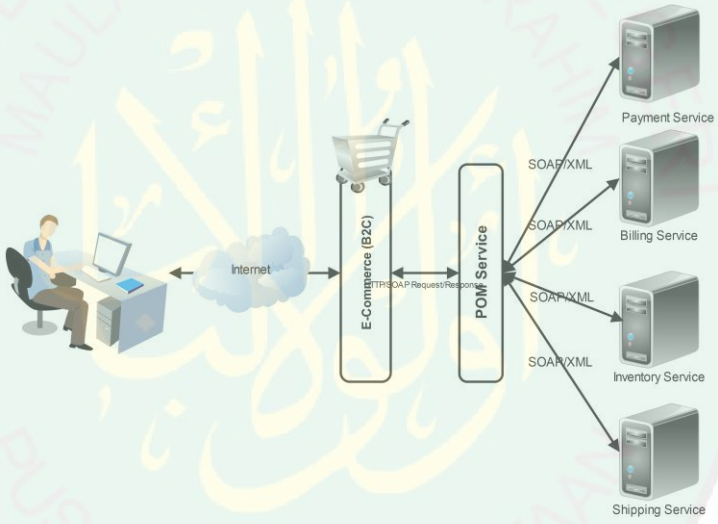
Pada layer ini menyediakan penyimpanan data dari semua pemrosesan yang terjadi pada *IT Section* sesuai dengan departemennya. *Database* dapat menyimpan informasi hasil transaksi yang terjadi mulai dari informasi produk, harga dan jumlah barang yang tersedia. Setiap departemen atau *IT Section* memiliki databasenya sendiri dengan jenis DBMS yang mungkin berbeda-beda.

4. *Network Infrastructure Layer*

Pada layer ini menyertakan komponen software yang bersifat *low level*. Jaringan telekomunikasi yang digunakan yaitu *protocol* dan *standard* seperti (XML,EDI) dan sistem operasi. Layer ini menyediakan antarmuka dengan perangkat jaringan dan fungsi-fungsinya seperti *routing, hosting* dan *control service.* Layer ini berfungsi sebagai perantara fisik pertukaran data dengan SOA nantinya.

3.2.2. Arsitektur SOA POM

Pada sistem *e-commerce* ini, sistem dirancang untuk dapat terintegrasi dengan sistem pada departemen yang lain untuk mendukung proses bisnis POM (*Purchase Order Managent*). Dalam sistem SOA POM, terdapat integrasi ke berbagai sistem dalam setiap departemen. Diantaranya integrasi sistem ke departemen gudang (*inventory*), pengiriman (*shipping*), penagihan (*billing*), dan pembayaran (*payment*). Arsitekur integrasi terdapat pada gambar 3.2 berikut ini.



Gambar 3.2. Model Arsitektur SOA

Basis data yang akan diakses kedalam sistem SOA POM pada masing-masing departemen diwakili oleh sebuah *web service* yang nantinya akan diolah atau diorkestrasi menjadi satu *web service* tunggal POM. Hasil *web service* dari POM kemudian diakses oleh aplikasi web *e-commerce*. Hasil yang ditampilakan adalah total tagihan *(invoice)* ketika proses POM selesai dijalankan.

3.3. Entitas *Web Service*

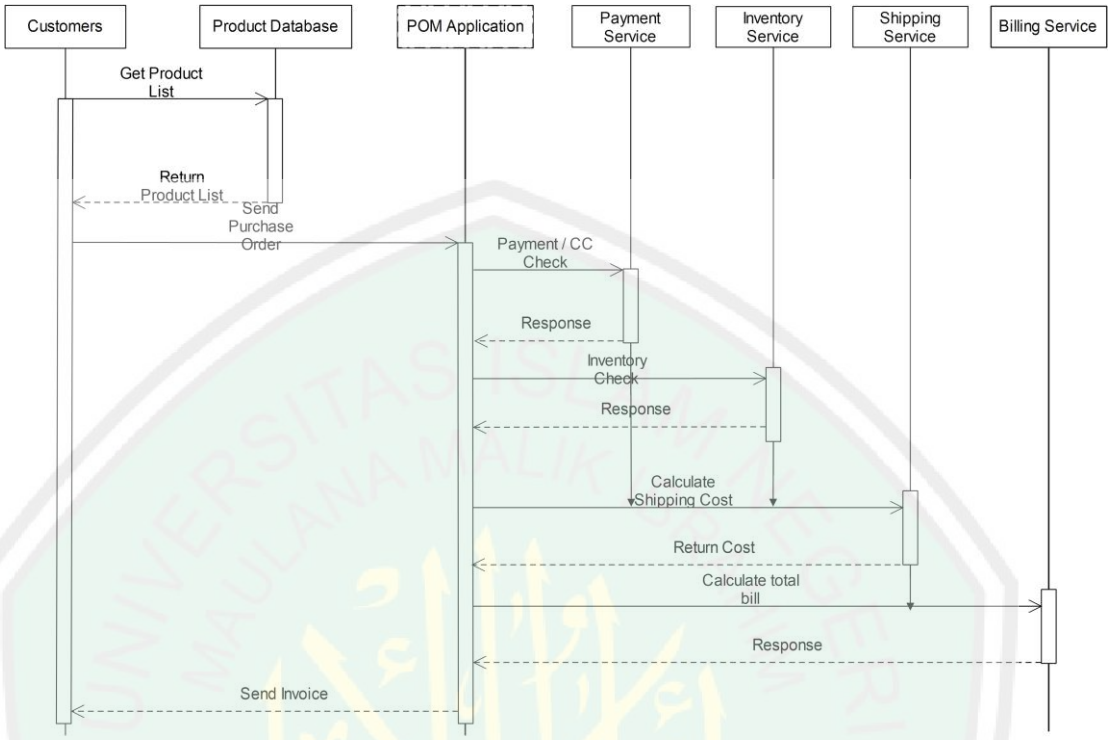
Berdasarkan arsitektur yang dibangun diatas kemudian didefinisikan peran serta letak dari masing-masing *web service* dalam setiap departemen yang ada. *Web service* merupakan komponen pembangun SOA. Dalam aplikasi POM terdapat komponen-komponen *web service* yang kemudian akan diorkestrasi melalui model *poin-to-point*. Penjelasan detail *web service* dapat dilihat pada tabel 3.1.

Tabel 3.1. POM *Web Service*

NO	IT Section	Nama Service	Sumber	Keterangan
1	*Sales*	*Payment Service*	Bank	Mengecek validasi kartu kredit, apakah masih aktif atau tidak.
2	*Inventory*	*Inventory Service*	Bagian Gudang	Cek ketersediaan barang di gudang
3	*Shipping*	*Shipping Service*	Perusahaan pengiriman barang	Cek biaya pengiriman
4	*Billing*	*Billing Service*	Bagian Penagihan	Mengkalkulasi total pembayaran dan mengirim *invoice* ke *customer*

3.4. *Sequence Diagram Web Service* POM

Setelah semua entitas *web service* tersedia selanjutnya adalah mengelola *web service* tersebut berdasarkan tujuan bisnis tertentu dalam hal ini POM. POM melibatkan sekumpulan *web service* diatas untuk diolah sesuai dengan proses alur bisnis POM itu sendiri. Alur pengelolaan *service* tersebut sebagaimana tertuang dalam sequence diagram POM pada gambar 3.3 dibawah ini.

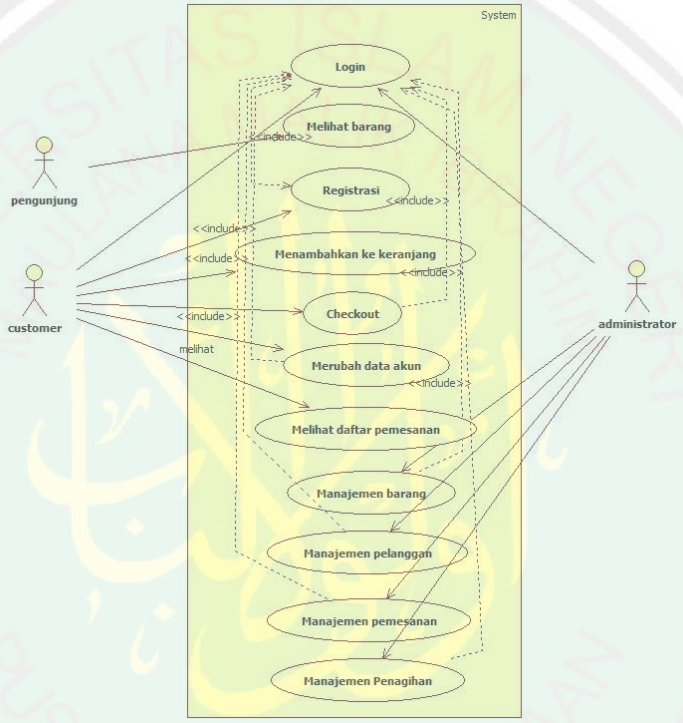


Gambar 3.3. POM Web Service (Baraka & Al-Ashqar, 2013)

Proses pertama terjadi pada saat *customer* melakukan *request purchase order* kepada POM *Application*. Kemudian *request* diteruskan kepada *payment service* yang berfungsi mengecek masa aktif kartu dan jumlah kredit kartu masih mencukupi apa tidak, jika valid maka akan dilanjutkan dengan pengecekan barang. Apakah barang dengan jumlah yang dipesan masih tersedia atau tidak, jika valid atau tersedia maka akan dilanjutkan ke proses cek biaya pengiriman barang dengan *response* biaya pengiriman barang. Kemudian setelah proses tersebut selesai maka akan dikalkulasi total tagihan dan customer akan menerima *invoice*.

3.2.3. *Use Case Diagram*

Diagram *use case* menggambarkan interaksi antara *stakeholder* dengan sistem yang dijalankan berdasarkan proses bisnis yang telah dijelaskan pada analisis sistem diatas. Seperti yang digambarkan pada gambar 3.4. berikut ini.



Gambar 3.4. Diagram *use case* sistem *e-commerce*

Deskripsi penjelasan aktor pada sistem *e-commerce* terintegrasi dapat dilihat pada tabel 3.2. di bawah ini.

Tabel 3.2. Definisi aktor pada diagram *use case*

No	Aktor	Deskripsi
1	Pengunjung	Orang yang mengunjungi situs dan hanya melihat barang-barang yang ada pada situs.
2	*Customer*/Pelangggan	Orang yang membeli barang pada situs *E-Commerce*.

3	*Administrator*	Orang yang bertugas dan memiliki hak akses untuk melakukan pengelolaan sistem *back end* secara keseluruhan.

Selanjutnya, pendefinisian *use case* dijelaskan secara detail pada tabel 3.3.

Tabel 3.3. *Definisi* use case

No	Nama *Use Case*	Deksripsi
1	Melihat barang	Menampilkan halaman katalog barang. Pelanggan dapat melihat jenis barang berdasarkan kategori tertentu beserta detail barang tersebut.
2	Registrasi	Menampilkan halaman dan form untuk registrasi pelanggan
3	Login	Menampilkan halaman dan form untuk *login* pelanggan
4	Menambah ke keranjang	Menambahkan barang ke keranjang, menampilkan halaman dan form pengelolaan keranjang.
5	Checkout	Menampilkan halaman dan form untuk melakukan *checkout* yang terdiri dari form untuk pembayaran (*payment*) serta form untuk detail pengiriman barang (*shipping*).
6	Merubah data akun	Menampilkan halaman dan form untuk merubah data personal dari akun pelanggan.
7	Melihat daftar pesanan	Menampilkan halaman dan *list* daftar pemesanan pelanggan
8	Manajemen pelanggan	Menampilkan halaman dan form untuk mengelola data pelanggan.
9	Manajemen pemesanan	Menampilkan halaman dan form untuk mengelola data pemesanan.

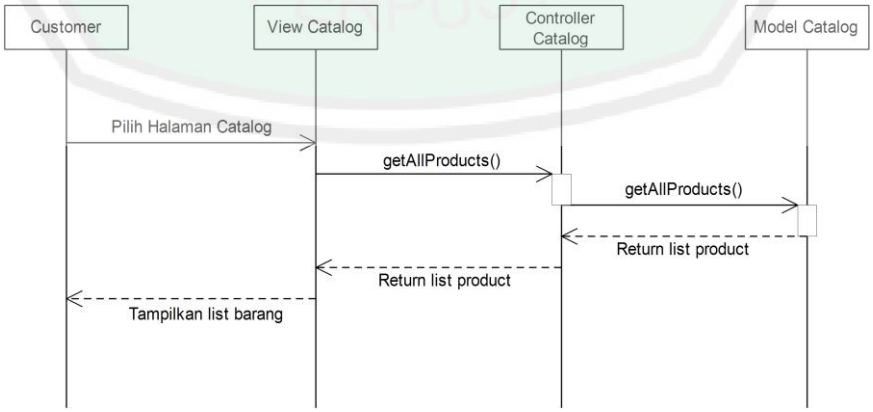
10	Manajemen barang	Menampilkan halaman dan form untuk mengelola data barang
11	Manajemen Penagihan	Menampilkan halaman penagihan dan form untuk mengelola penagihan.

3.2.4. *Sequence Diagram*

Diagram *sequence* digunakan untuk menggambarkan proses pengiriman dan penerimaan pesan yang terjadi dalam *use case*. Banyaknya diagram sequence yang dibangun adalah sama dengan banyaknya use case yang ada. Berikut adalah diagram *sequence* yang dibangun:

1. *Sequence diagram* melihat barang

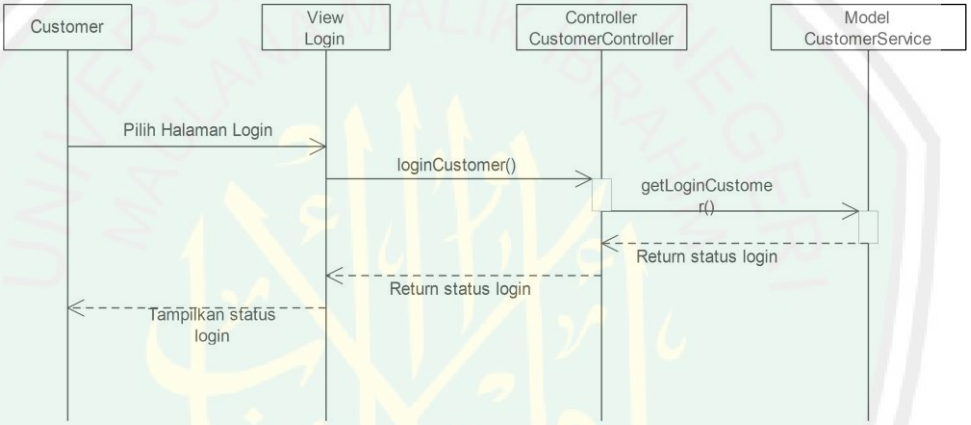
Pada gambar 3.5. dijelaskan jika seorang *customer* memilih menu katalog maka akan memanggil fungsi *getAllProducts()* pada *class ProductController* kemudian dari *controller* ini akan memanggil fungsi *getAllProducts()* model pada *ProductService* kemudian hasilnya dikembalikan ke *controller* dan ditampilkan di view katalog.



Gambar 3.5. Diagram *sequence* melihat barang

2. *Sequence diagram login*

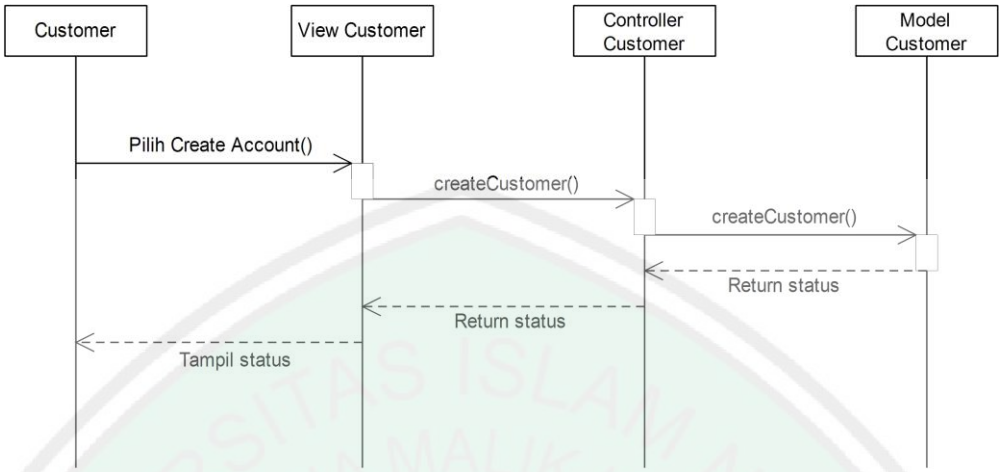
Pada gambar 3.6. dijelaskan bahwa customer melakukan login pada halaman login dengan memanggil fungsi loginCustomer() pada class CustomerController dan kemudian memanggil fungsi getLoginCustomer() pada class CustomerService. Apabila login sukses maka akan muncul status sukses dan customer akan menuju halaman akunnnya.



Gambar 3.6. Diagram *sequence login*

3. *Sequence diagram* registrasi

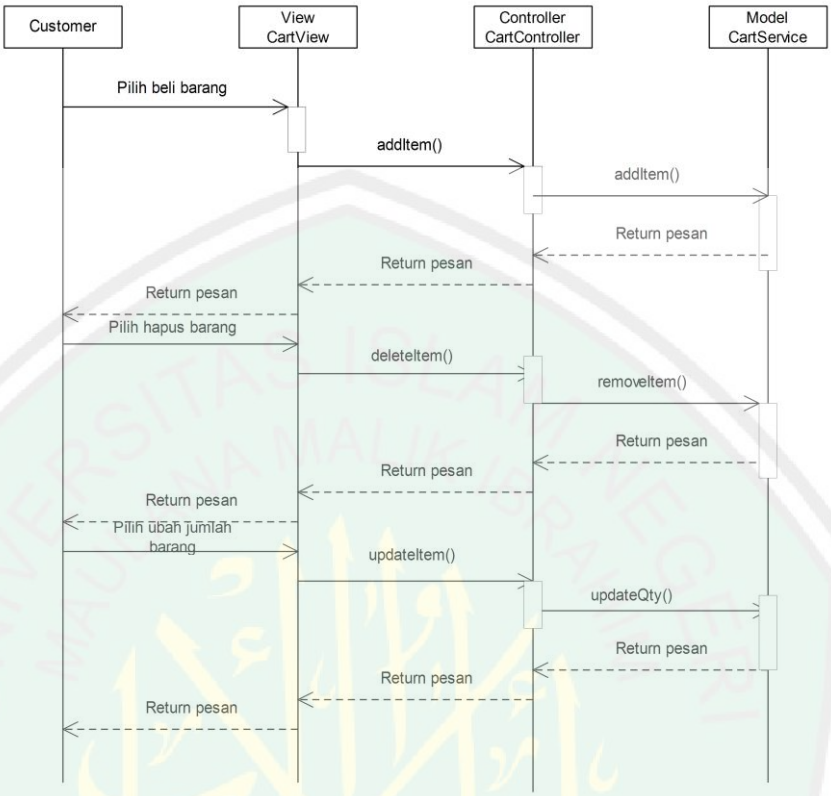
Pada gambar 3.7. dijelaskan bahwa *customer* melakukan pemilihan menu registrasi pada situs. Kemudian masuk ke halaman registrasi dengan nama *ViewCustomer* kemudian saat proses registrasi memanggil *createCustomer()* pada *CustomerController* dan menyimpan data pada *CustomerService* melalui *method createCustomer().*



Gambar 3.7. Diagram *sequence* registrasi

4. *Sequence diagram* memasukkan ke keranjang

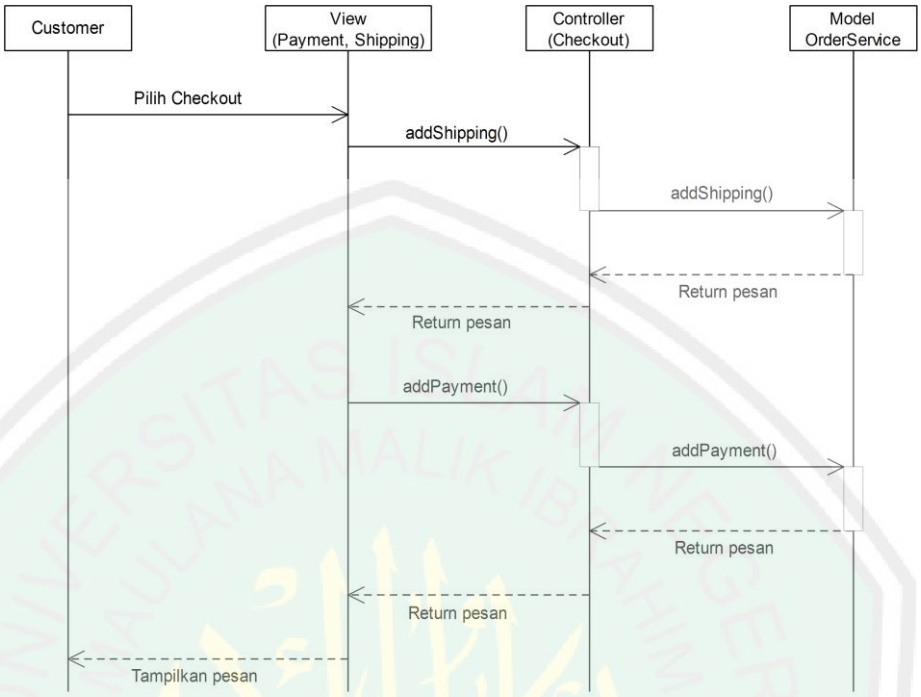
Pada gambar 3.8 dijelaskan bahwa *customer* mengklik tombol beli pada barang tertentu, maka barang akan dimasukkan kedalam keranjang belanja. saat memasukkan barang ke keranjang akan memanggil fungsi *addItem()*. Kemudian dari CartController akan memanggil fungsi *addItem()* pada CartService. Proses selanjutnya adalah hapus dan ubah jumlah barang pada keranjang. Yang masing-masing akan memanggil fungsi *deleteItem()* dan *updateItem()* pada *CartController* yang terhubung dengan model *CartService* dengan fungsi *removeItem()* dan *updateQty*.



Gambar 3.8. Diagram *sequence* memasukkan ke keranjang

5. *Sequence diagram* checkout

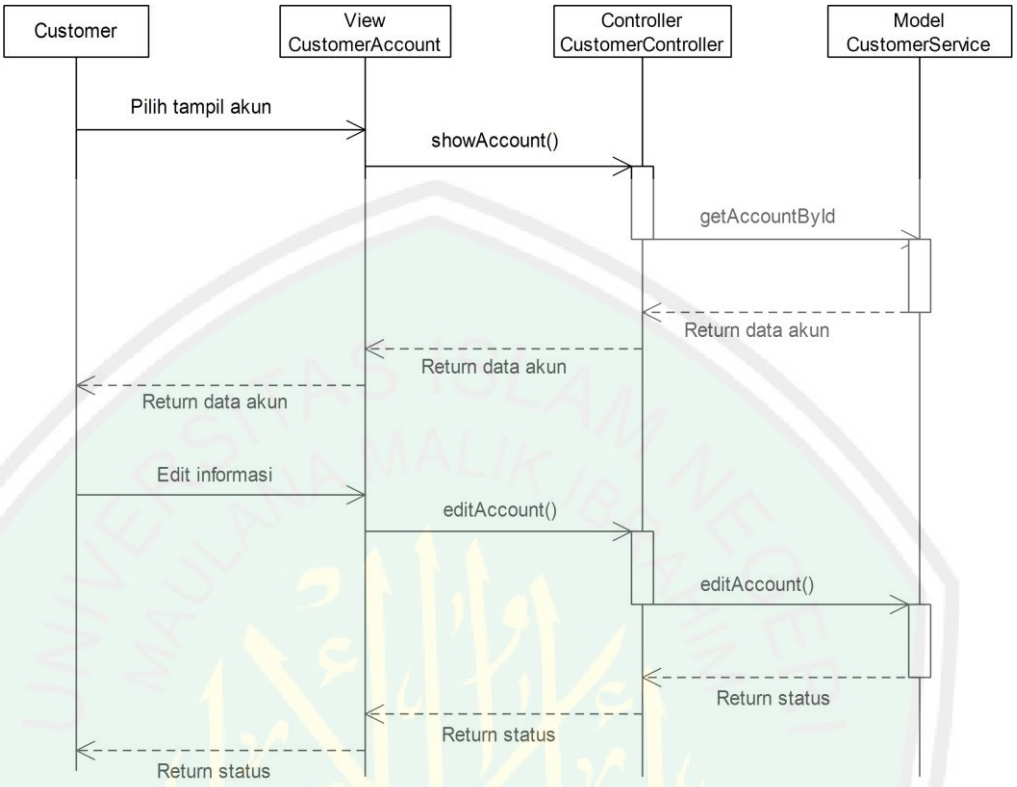
Pada gambar 3.9. dijelaskan bahwa *customer* memilih tombol lanjutkan ke pembayaran maka akan masuk ke proses *checkout*. Dalam proses *checkout* terdapat dua sub proses yakni *addShipping()* dan *addPayment()* dimana kedua fungsi tersebut terdapat pada *CheckoutController*. Pada fungsi *addShipping()* akan menyimpan data melalui *OrderService*. Apabila penyimpanan sukses maka akan menuju halaman metode pembayaran dengan memanggil fungsi *addPayment()*. Setelah diisi metode pembayaran yang digunakan maka data akan disimpan melalui *OrderService* dengan fungsi *addPayment()*. Barulah setelah dua proses ini selesai maka akan menampilkan pesan *checkout* sukses.



Gambar 3.9. Diagram *sequence checkout*

6. *Sequence diagram* merubah data akun (*change account*)

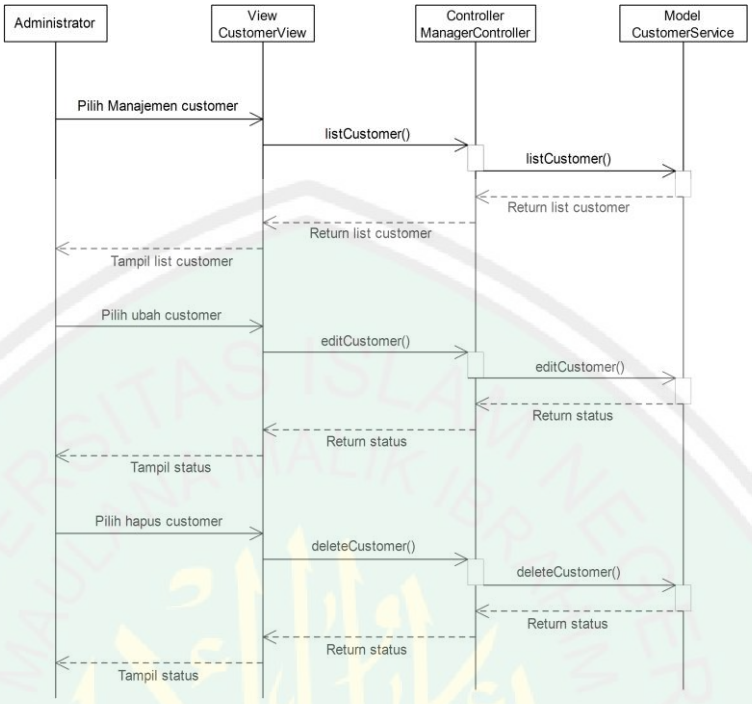
Pada gambar 3.10. dijelaskan bahwa jika *customer* setelah melakukan *login* dapat melihat detail akunnya dan bisa melakukan edit informasi akunnya. Informasi tersebut terkait detail akun. Jika kita memilih tampil akun maka akan diproses oleh fungsi *showAccount()* pada *controller* yang selanjutnya akan memanggil fungsi pada model yakni *getAccountById()* dan data akan dikembalikan ke *CustomerController* dan diforward ke *customer* tentang detail akun yang dimilikinya.



Gambar 3.10. Diagram *sequence* merubah data akun

7. *Sequence diagram* manajemen pelanggan (*customer manager*)

Pada gambar 3.11. dijelaskan bahwa jika seorang admin memilih menu manajemen pelanggan, maka sistem akan memanggil fungsi *listCustomer()* pada *class ManagerController*. Selanjutnya memanggil data *customer* yang ada pada basis data dengan memanggil fungsi *listCustomer()* pada *class CustomerService*. Apabila semua proses berhasil makan menampilkan data *listCustomer()* pada administrator. Selanjutnya jika administrator memilih menu *editCustomer()* maka sistem akan memanggil fungsi *editCustomer()* berdasarkan id tertentu dan data hasil ubahan akan disimpan kedalam database dengan memanggil fungsi *editCustomer()* pada *class CustomerService*. Apabila proses edit data berhasil maka akan menampilkan pesan bahwa data berhasil *edit*.

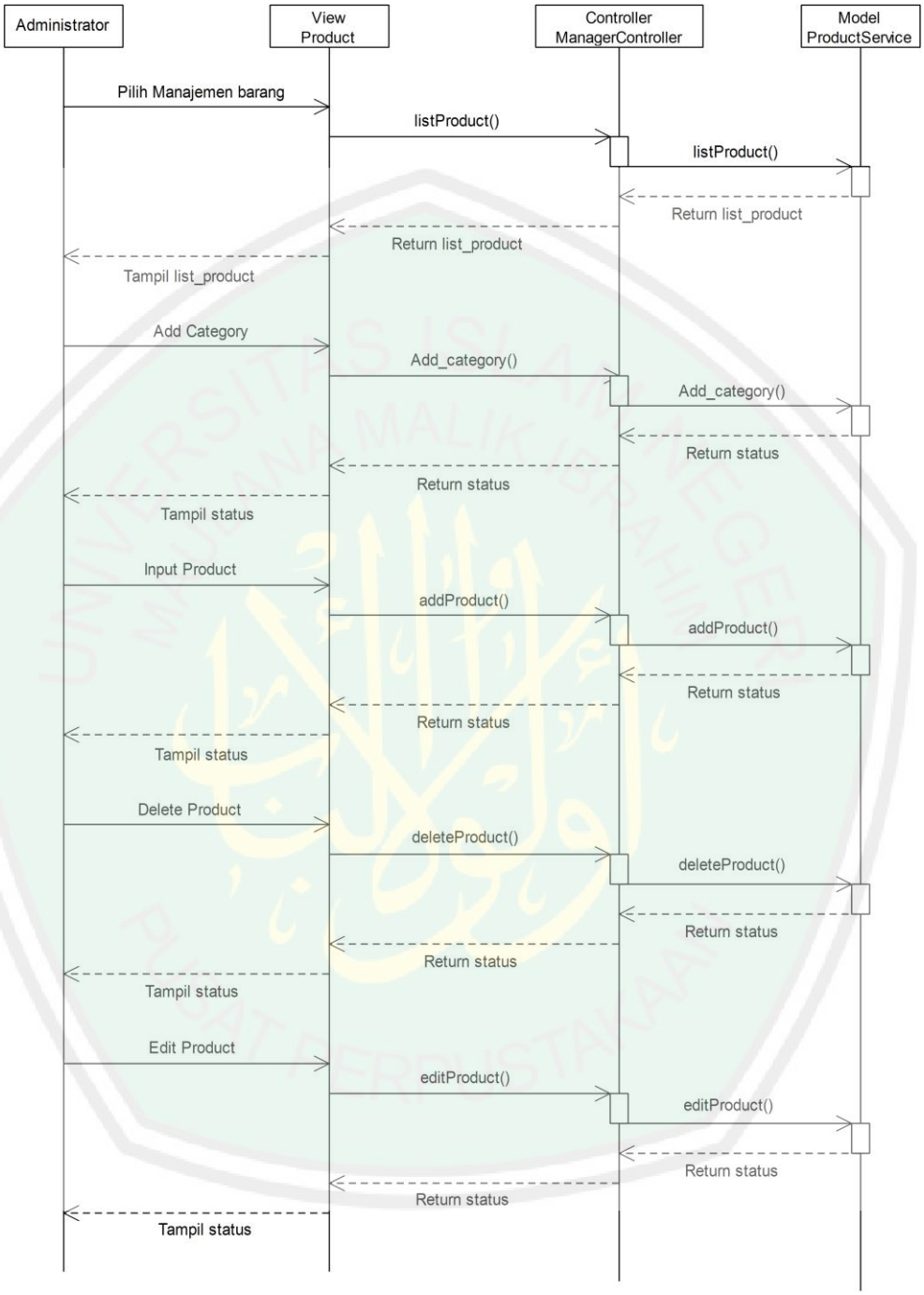


Gambar 3.11. Diagram sequence manajemen pelanggan

8. *Sequence diagram* manajemen barang

Pada gambar 3.12. dijelaskan jika seorang *administrator* memilih menu manajemen katalog, maka akan memanggil fungsi *listProduct()* pada *class ManagerController* dan selanjutnya mengambil data dengan memanggil fungsi *listProduct()* pada *class ProductService*. Setelah berhasil maka akan menampilkan daftar produk yang ada. Kemudian jika administrator memilih *input product* maka sistem akan memanggil *addProduct()* dan *addCategory()* fungsi pada *class ManagerController* dan selanjutnya menyimpan data dengan memanggil fungsi *addProduct()* dan *addCategory()* pada *class model ProductService*. Apabila proses penyimpanan data berhasil maka akan menampilkan pesan berhasil input data. Jika administrator memilih hapus produk, maka sistem akan memanggil fungsi *deleteProduct()* pada *class*

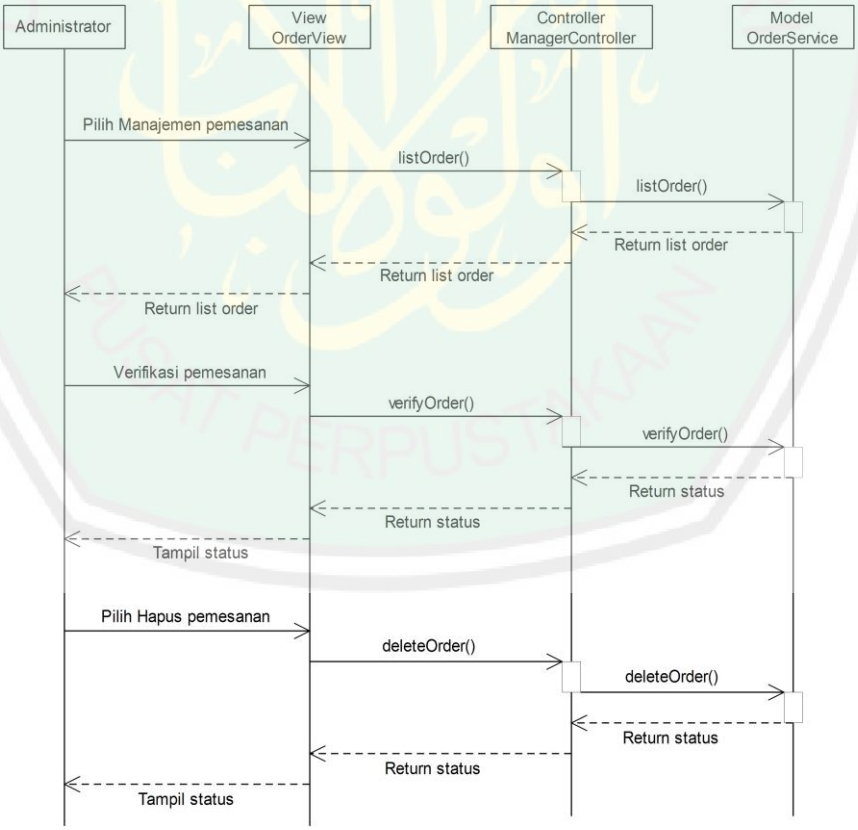
ManagerController dan selanjutnya menghapus data dengan memanggil fungsi *deleteProduct()* pada *class ProductService*.. Apabila proses hapus data berhasil maka akan menampilkan pesan berhasil hapus data. Jika administrator memilih edit produk, maka sistem akan memanggil fungsi *editProduct()* pada *class ManagerController* dan selanjutnya mengubah data dengan memanggil fungsi *editProduct()* pada *class ProductService*. Apabila proses ubah data berhasil maka akan menampilkan pesan berhasil ubah data.



Gambar 3.12. Diagram *sequence* manajemen barang

9. *Sequence diagram* manajemen pemesanan (*order manager*)

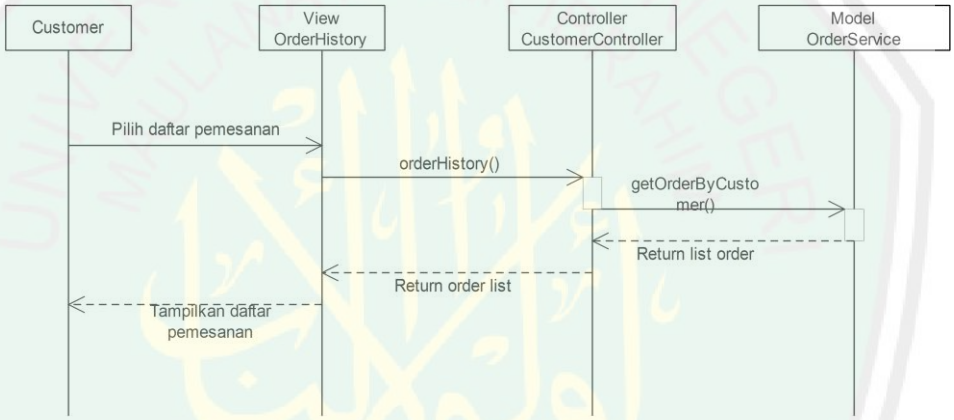
Pada gambar 3.13. dijelaskan bahwa jika administrator memilih manajemen pemesanan maka sistem akan memanggil fungsi *listOrder()* pada *class ManagerController* dan selanjutnya mengambil list data pemesanan dengan memanggil fungsi *listOrder()* pada *class OrderService*. Administrator bisa melakukan verifikasi order dengan memanggil method verifyOrder() dan melakukan hapus atau pembatalan order dengan memanggil method *deleteOrder()*.



Gambar 3.13. Diagram *sequence* manajemen pemesanan

10. *Sequence diagram* melihat daftar pemesanan

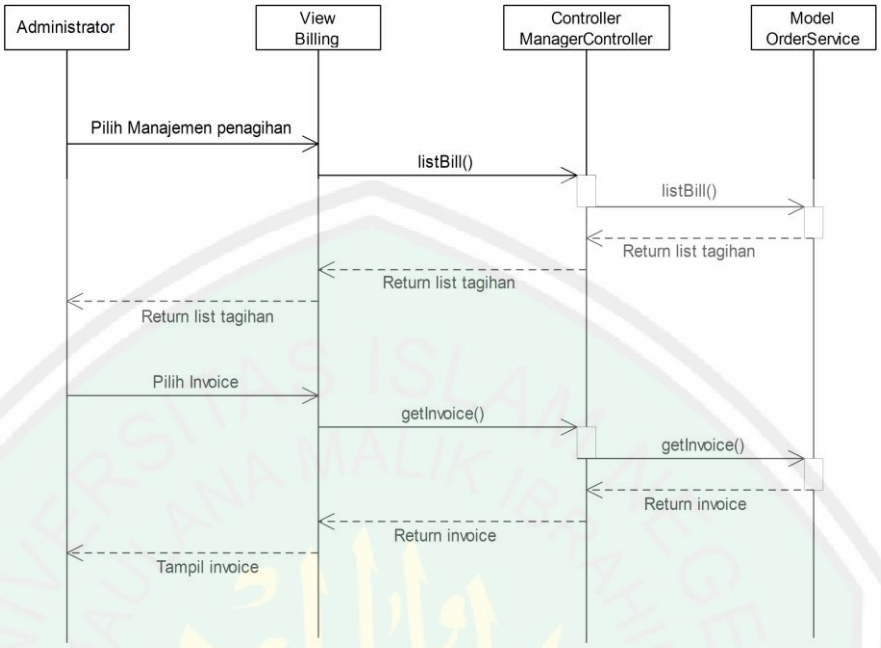
Pada gambar 3.14. dijelaskan bahwa jika pelanggan memilih daftar pemesanan. Kemudian memanggil fungsi *orderHistory()* pada *CustomerController* dan diteruskan pada *class model OrderService* dengan memanggil fungsi *getOrderByCustomer().* Kemudian menampilkan daftar pemesanan pada pelanggan tersebut.



Gambar 3.14. Diagram *sequence* melihat daftar pemesanan

11. *Sequence diagram* manajemen penagihan

Pada gambar 3.15. dijelaskan bahwa administrator memilih manajemen penagihan. Kemudian memanggil fungsi *listBill()* pada *ManagerController* dan diteruskan pada *class model OrderService* dengan memanggil fungsi *listBill().* Kemudian menampilkan daftar pemesanan pada pelanggan tersebut. Selain itu terdapat proses invoice dengan memanggil fungsi *getInvoce()* pada class *ManagerContoller* dan *OrderService.*



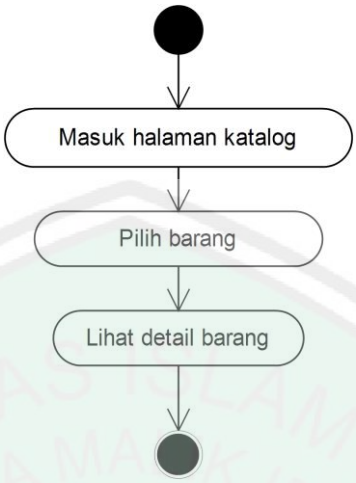
Gambar 3.15. Diagram *sequence* manajemen penagihan

3.2.5. *Activity Diagram*

Activity diagram digunakan untuk menggambarkan aktivitas yang terjadi pada sebuah sistem. Tujuannya yaitu memodelkan aspek dinamis yang berjalan pada sebuah sistem. Pada dasarnya *sequence diagram* hampir sama dengan flowchart, akan tetapi dalam paradigma permodelan dengan UML model inilah yang dipakai. Adapun *activity diagram* sistem *e-commerce* yakni sebagai berikut:

1. Diagram *activity* melihat barang dikatalog

Gambar 3.16. menunjukkan alur sistem pada fungsi melihat barang di katalog. Ketika kita masuk ke katalog barang maka akan terdapat beberapa barang, kita bisa memilih tombol lihat detail barang untuk melihat detail barang yang tersedia.



Gambar 3.16. Diagram *activity* melihat barang di katalog

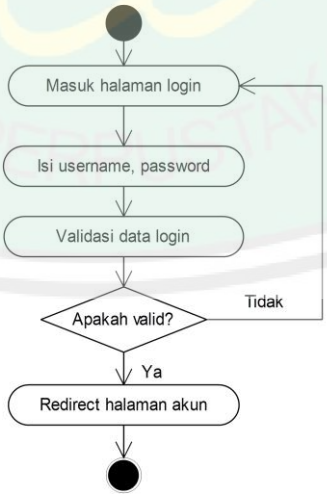
2. Diagram *activity* registrasi

Gambar 3.17. menunjukkan alur sistem pada registrasi user. Ketika calon pelanggan melakukan register maka akan muncul form registrasi, setelah itu user akan memasukkan data terkait dengan registrasi sampai kemudian di-*submit*. Setelah melakukan *submit* maka data akan divalidasi, apakah data tersebut telah sesuai dengan format data pada database atau tidak. Setelah validasi selesai dan data benar-benar valid maka proses selanjutnya adalah menyimpan data pada database dan registrasi berjalan sukses.

Gambar 3.17. Diagram *activity* registrasi

3. Diagram *activity login*

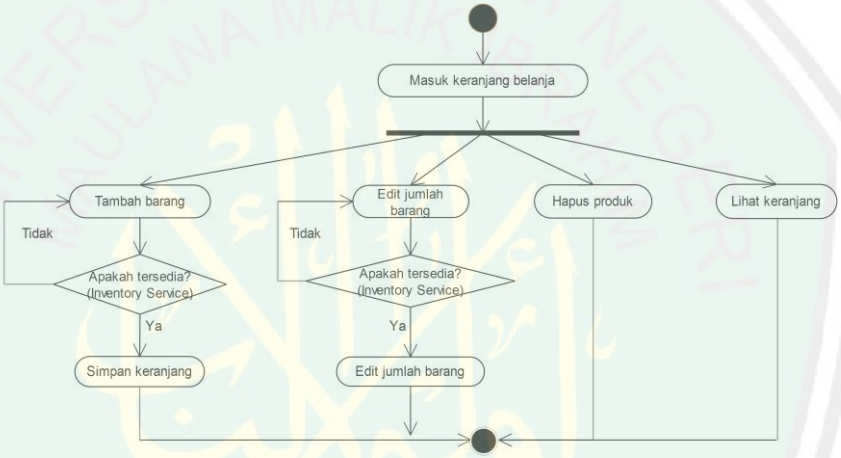
Gambar 3.18. menunjukkan proses login. Pelanggan harus memasukkan data username dan password. Setelah itu akan dilakukan pengecekan login dalam database. Apabila login valid maka pelanggan akan diredirect ke halaman akun.



Gambar 3.18. Diagram *activity login*

4. Diagram *activity* memasukkan barang ke keranjang

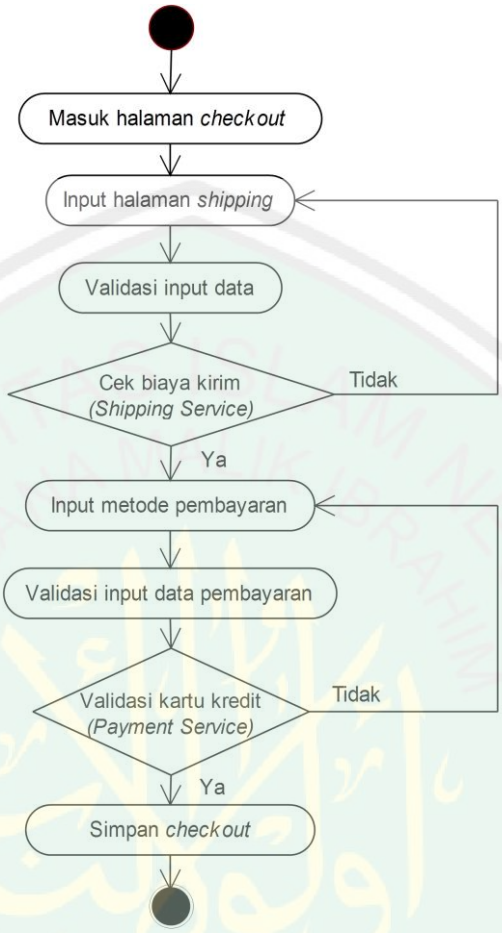
Gambar 3.19. menunjukkan proses memasukkan barang kedalam keranjang. Proses yang terjadi dalam activity ini adalah tambah barang, edit jumlag barang, hapus barang, dan melihat keranjang. Dalam tambah barang dan edit barang terdapat pengecekan apakah item barang dengan jumlah tertentu tersedia di gudang. Apabila tersedia maka akan diproses.



Gambar 3.19. Diagram *activity* memasukkan ke keranjang

5. Diagram *activity* checkout

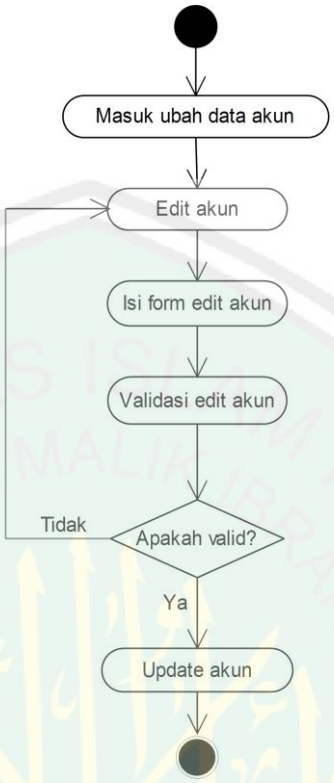
Gambar 3.20. menunjukkan alur proses *checkout*. Ketika user sudah memilih barang apa yang akan dipesan maka proses selanjutnya adalah melakukan pengisian alamat pengiriman dengan metode pembayaran yang digunakan. Proses selanjutnya adalah validasi data inputan. Jika semua data valid maka proses order akan diproses.



Gambar 3.20. Diagram *activity checkout*

6. Diagram *activity* merubah data akun

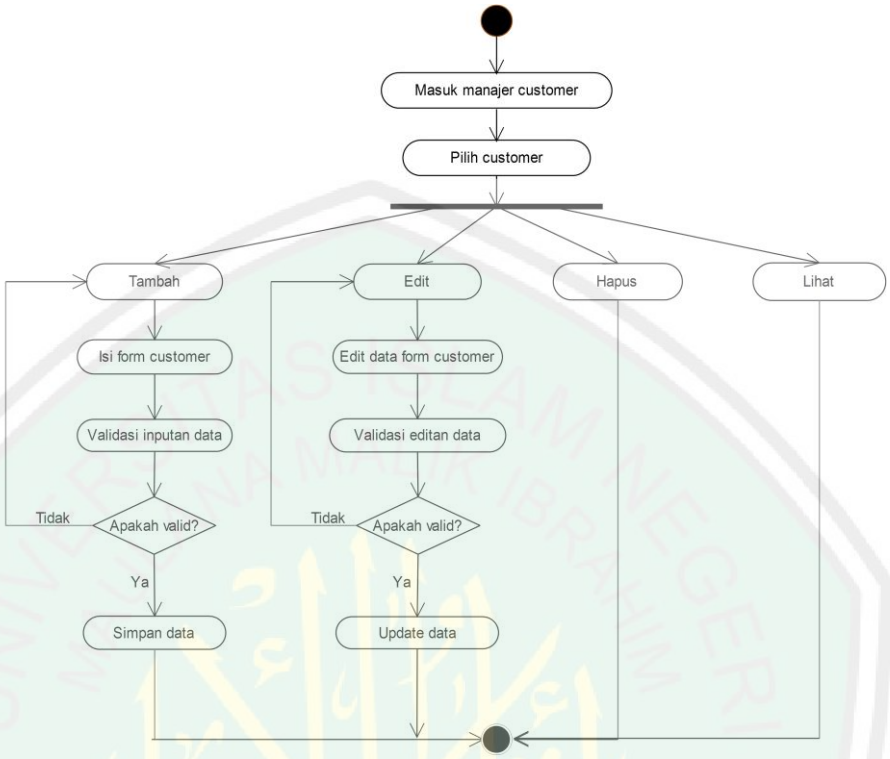
Gambar 3.21. Menunjukkan alur sistem pada fungsi merubah data akun. Manajemen akun berfungsi untuk mengatur informasi akun masing-masing *customer*. Pelanggan bisa mengubah informasi biodata diri dan lain sebagainya.



Gambar 3.21. Diagram *activity* merubah data akun

7. Diagram *activity* manajemen pelanggan (*customer*)

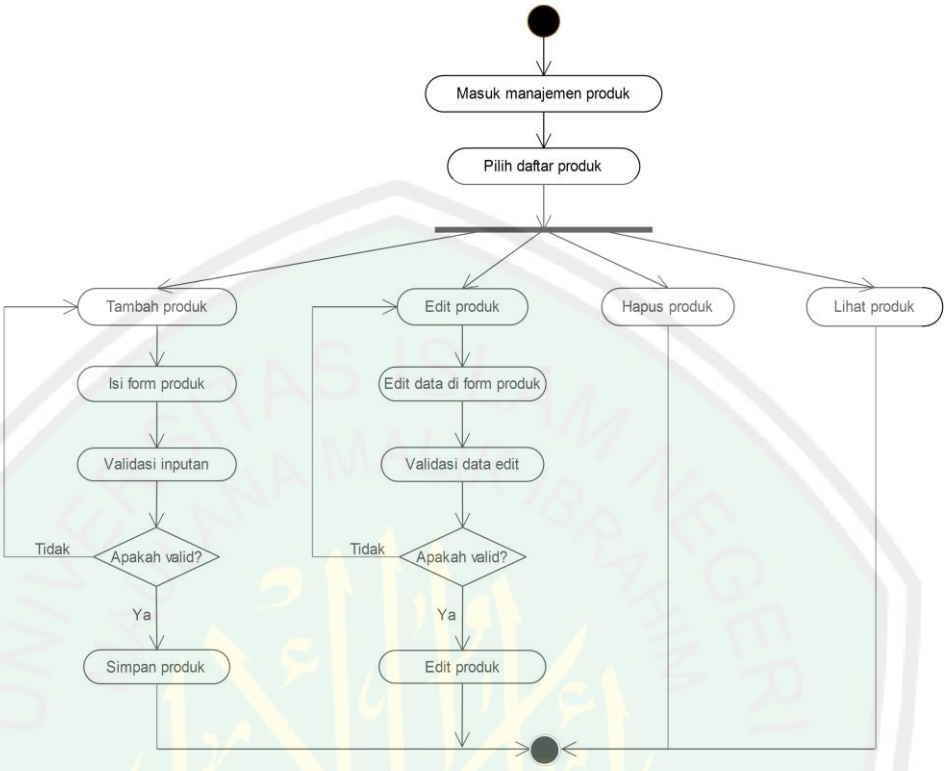
Gambar 3.22 menunjukkan alur sistem pada fungsi manajemen pelanggan. Ketika admin masuk ke halaman *manajer customer* maka akan melihat siapa saja *customer* yang sudah mendaftar. Admin memiliki control hak akses penuh untuk semua customer yakni bisa menambah, hapus, mengubah dan melihat detail pelanggan tersebut.



Gambar 3.22. Diagram *activity* manajemen pelanggan

8. Diagram *activity* manajemen barang

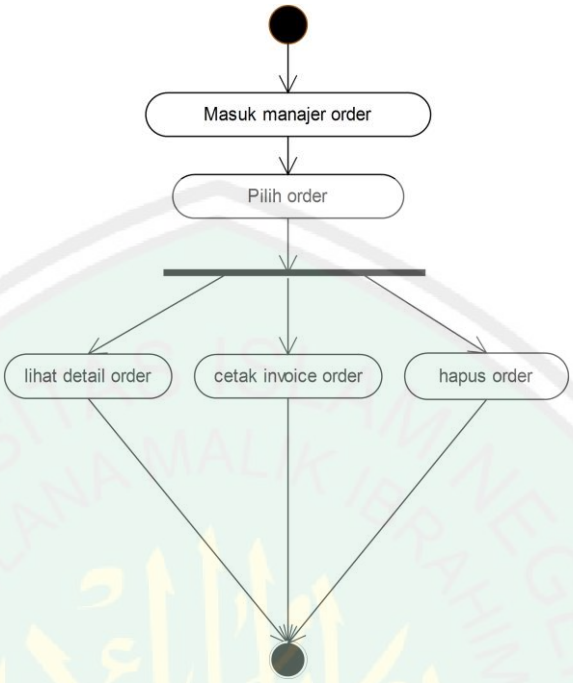
Gambar 3.23. menunjukkan alur sistem pada fungsi manajemen katalog. Ketika admin melihat menu manajemen katalog maka di sana akan terlihat beberapa barang, admin mempunyai control penuh terhadap semua barang. Admin bisa menambah, mengubah, menghapus dan melihat detail barang yang ada.



Gambar 3.23. Diagram *activity* manajemen barang

9. Diagram *activity* manajemen pemesanan (*order*)

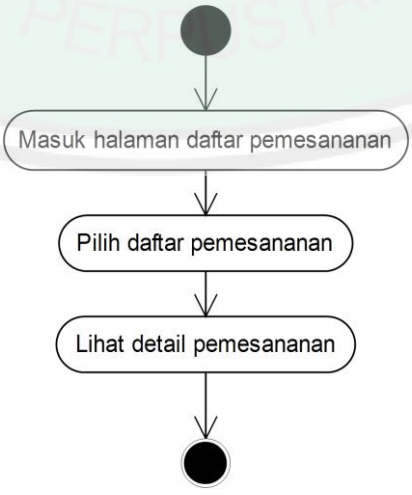
Gambar 3.24. menunjukkan alur sistem pada fungsi manajemen order. Fungsi manajemen *order* akan mengambil semua data order dan admin dapat meng-akses data *order* tersebut. Admin dapat melakukan dua proses, pertama proses verifikasi *order* yakni terkait pengecekan barang di *inventory*, pengecekan kartu kredit dan total pembayaran, dan dapat menonaktifkan *order* atau menghapusnya jika *order* tersebut tidak valid. Namun, jika order valid maka akan melakukan set jumlah barang pada gudang, dan mengurangi kredit pada kartu kredit. Yang kedua proses hapus yakni admin melakukan hapus *order*.



Gambar 3.24. Diagram *activity* manajemen pemesanan

10. Diagram *activity* melihat daftar pemesanan

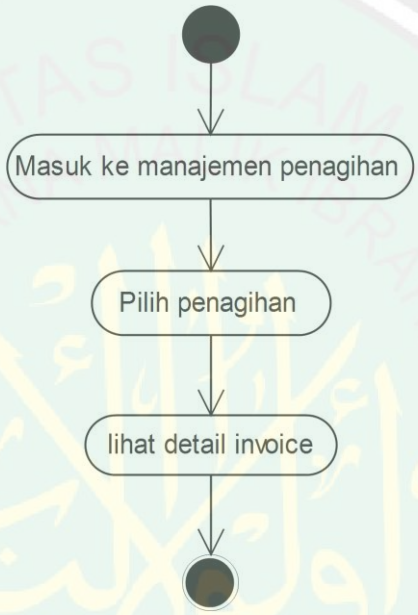
Gambar 3.25. menunjukkan alur sistem pada fungsi manajemen pemesanan. Pelanggan masuk ke halaman pemesanan, kemudian melihat detail dan status pemesanan tersebut dalam bentuk *invoice.*



Gambar 3.25. Diagram *activity* melihat daftar pemesanan

11. Diagram *activity* manajemen penagihan

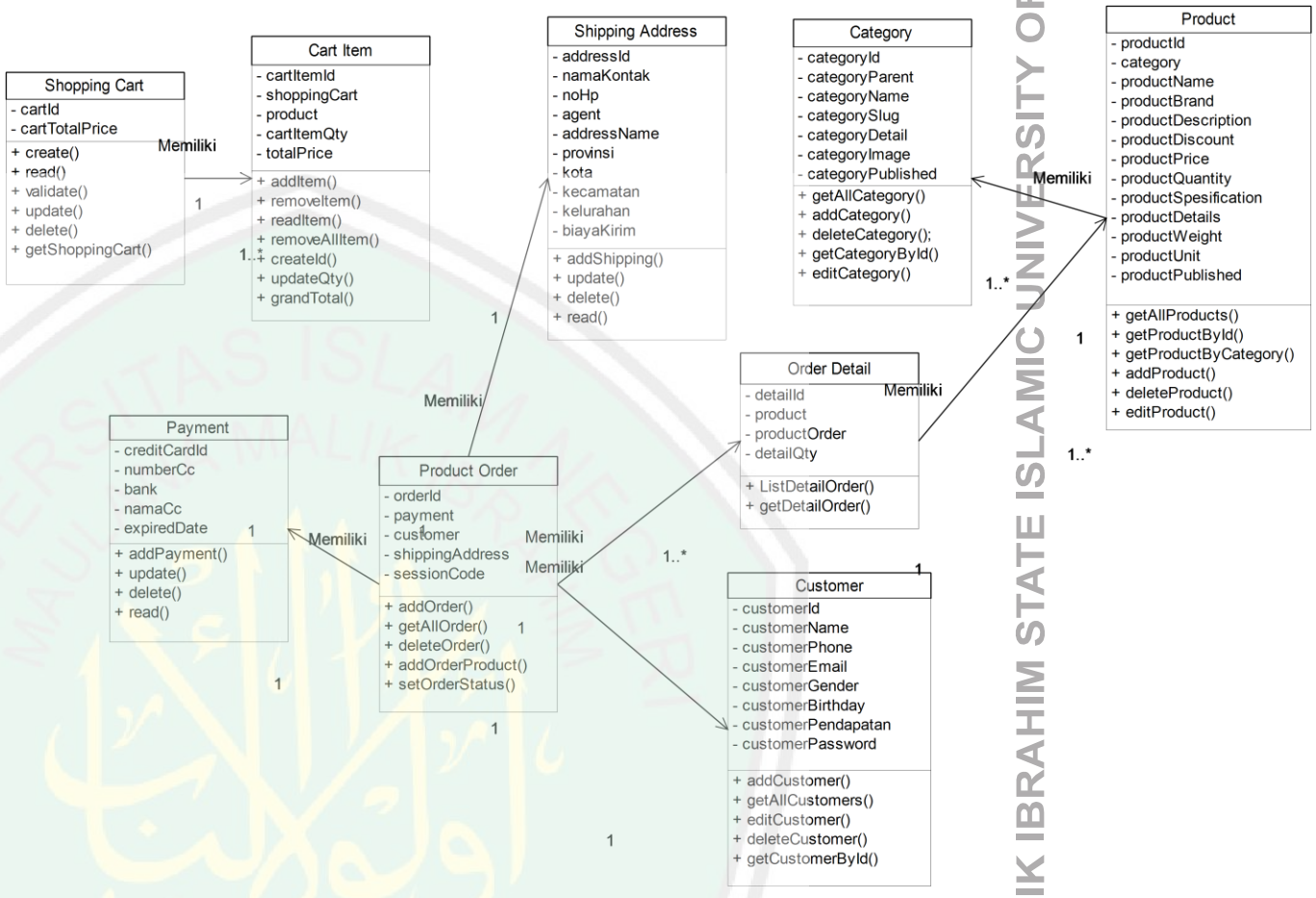
Gambar 3.26. menunjukkan alur sistem pada fungsi manajemen penagihan. Administrator masuk ke halaman manajemen penagihan dan melihat detail penagihan dalam bentuk invoice.



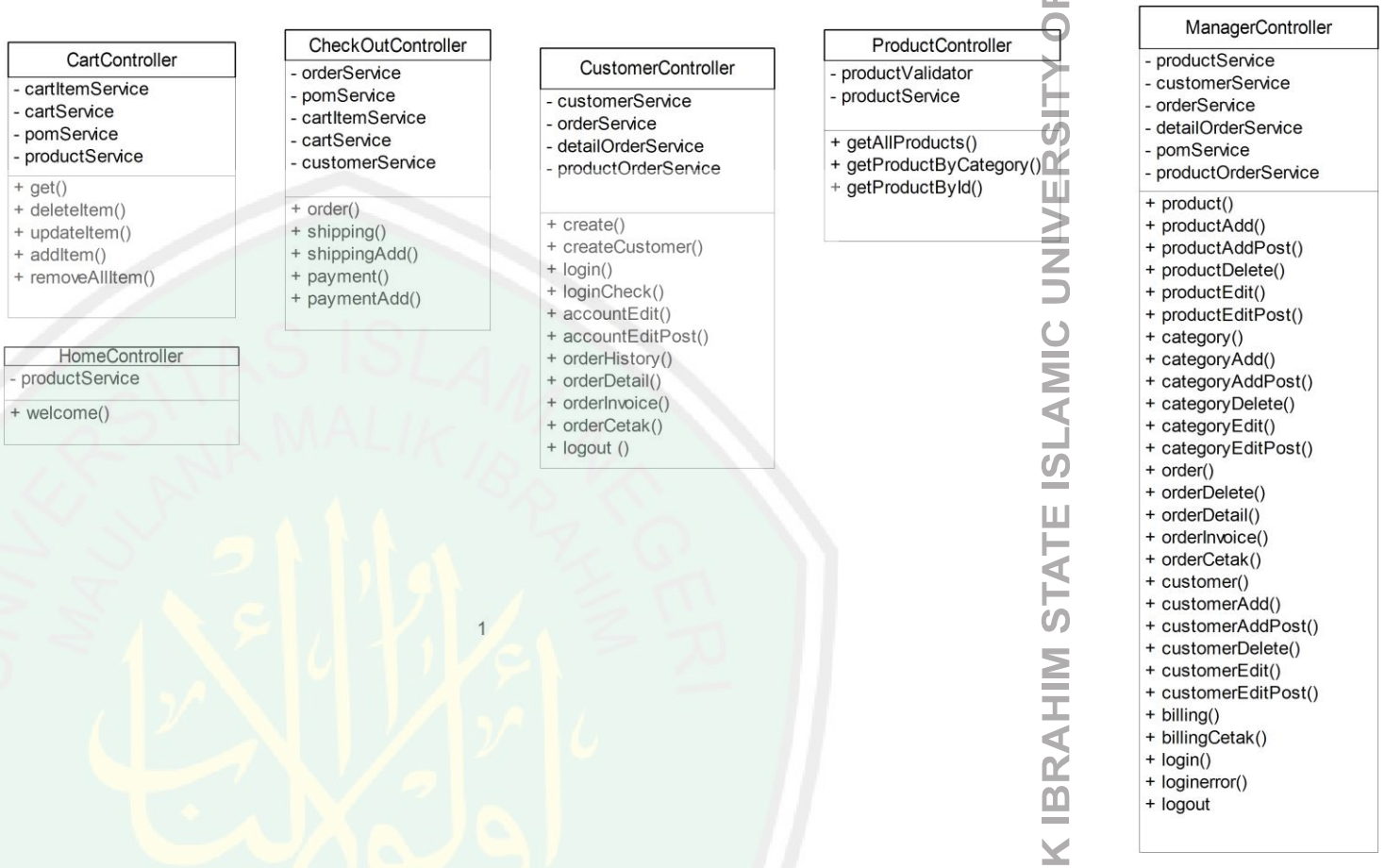
Gambar 3.26. Diagram *activity* manajemen penagihan

3.2.6. *Class Diagram*

Class diagram atau diagram kelas merupakan permodelan UML yang digunakan untuk memodelkan struktur sistem yang terbagi dalam kelas-kelas. Kelas memiliki *attribute* dan *method*. *Attribute* merupakan variable yang dimiliki oleh kelas, semetara *method* adalah fungsi-fungsi yang dimiliki oleh kelas. Dalam sistem ini memiliki beberapa kelas yang saling berkaitan satu sama lain, sebagai mana dijelaskan dalam class diagram berikut ini:



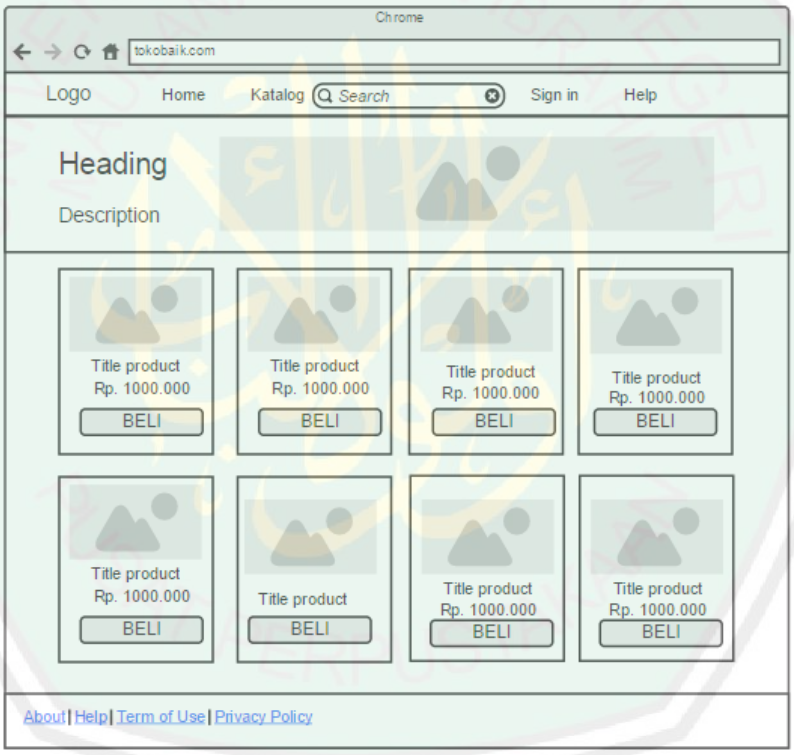
Gambar 3.27. Diagram *class e-commerce* (*repository layer* & *service layer*)



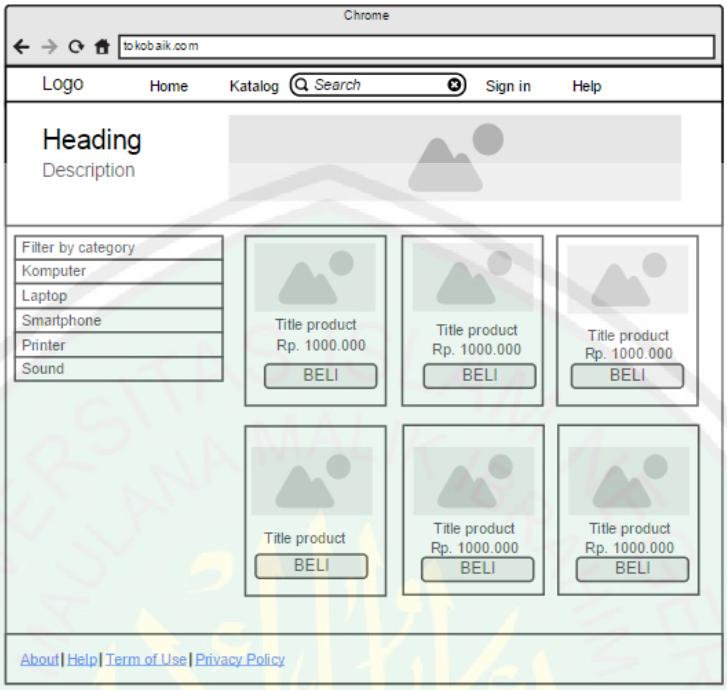
Gambar 3.28. Diagram *class controller e-commerce* (*presentation layer*)

3.2.7. Perancangan Antar Muka

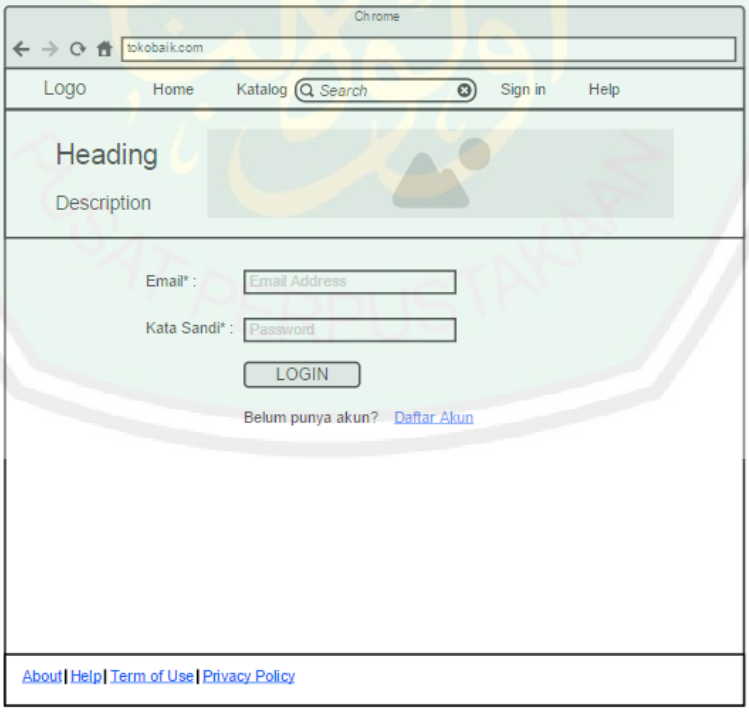
Pada bagian ini akan ditampilkan rancangan antarmuka *(interface)* aplikasi *e-commerce*. Beberapa tampilan diataranya adalah halaman index, halaman katalog, halaman login user dan halaman admin. Secara layout terdiri dari *navigation bar*, *header*, *content* dan *footer*. Yang berubah secara dinamis adalah konten mengikuti kebutuhan yang ada. Berikut desain tampilannya :



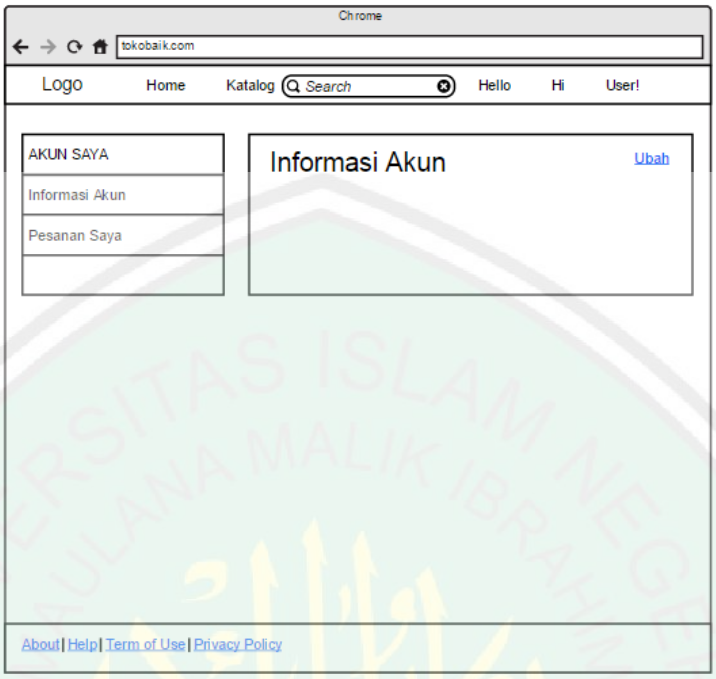
Gambar 3.29. Desain *interface* halaman utama



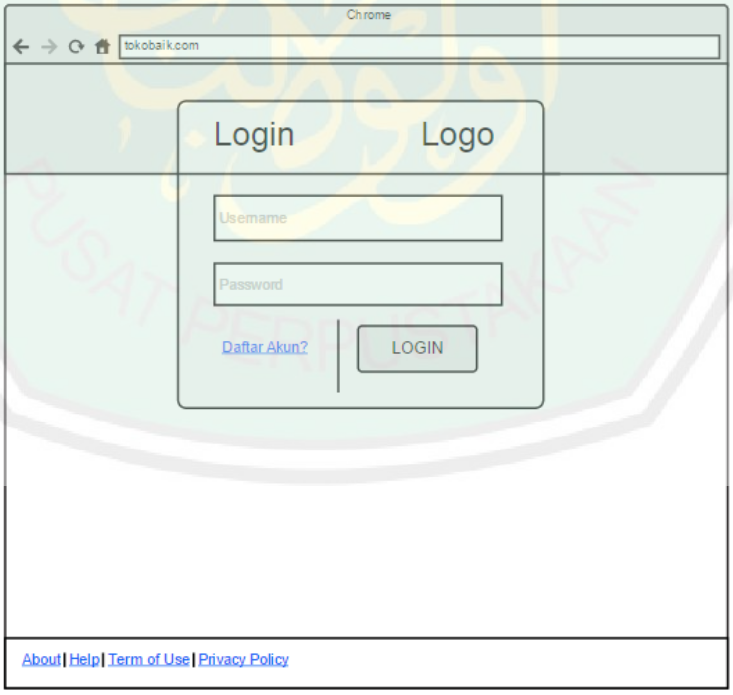
Gambar 3.30. Desain *interface* halaman katalog



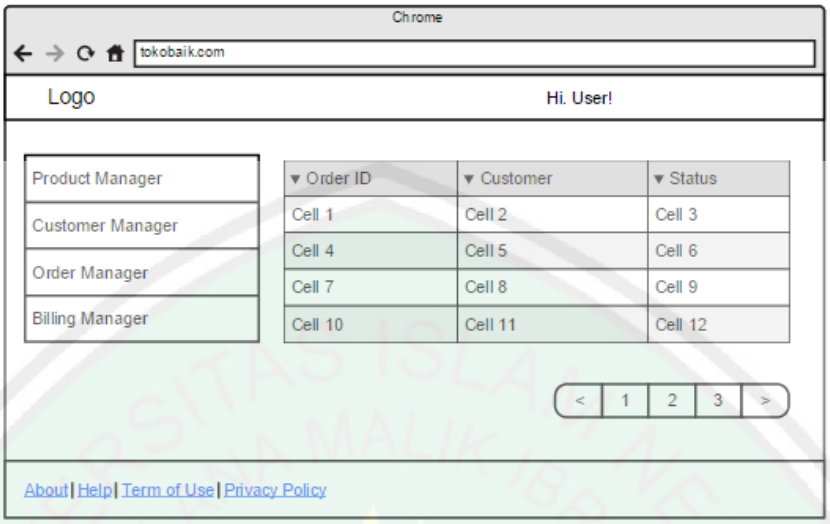
Gambar 3.31. Desain *interface* halaman *login customer*



Gambar 3.32. Desain *interface* halaman *dashboard user*



Gambar 3.33. Desain *interface* halaman *login admin*



Gambar 3.34. Desain *interface* halaman *administrator*

3.2.8. Perancangan Basis Data

Database yang digunakan untuk pengembangan sistem ini menggunakan DBMS MySQL. Dalam database yang digunakan terdiri dari 8 tabel yang terdiri dari tabel *product*, tabel *category*, tabel *cart_item*, tabel *shopping_cart*, tabel *product_order*, tabel *shipping_address*, tabel *payment*, dan tabel *customer*.

Tabel *product* memiliki 15 *field* dengan *product_id* sebagai *primary key*. Detail tabel *customer* secara lengkap disajikan pada tabel 3.4

Tabel 3.4. Desain tabel *product*

Nama Kolom	Tipe Data	Lebar	Keterangan
product_sku	integer	11	*primary key*, *sku (stock keeping unit)*
category_id	integer	11	*foreign key category*
product_name	varchar	300	nama produk
product_brand	varchar	255	brand produk
product_description	text		deskripsi singkat produk
product_discount	integer	11	diskon untuk produk
product_price	integer	11	harga untuk produk
product_credit	integer	11	harga cicilan untuk produk

credit_quantity	integer	11	banyaknya cicilan yang dilakukan
product_spesification	text		spesifikasi untuk produk
product_details	text		detail uraian lengkap produk
product_weight	varchar	30	berat barang
product_unit	varchar	100	unit
product_published	integer	11	status barang ditampilkan atau tidak

Tabel *category* memiliki 7 *field* dengan category_id sebagai *primary key*. Detail tabel *category* secara lengkap disajikan pada tabel 3.5.

Tabel 3.5. Desain tabel *category*

Nama Kolom	Tipe Data	Lebar	Keterangan
category_id	integer	11	*primary key*
category_parent	integer	11	kategori induk untuk sub kategori
category_name	varchar	300	nama kategori
category_slug	varchar	300	*slug url* untuk kategori
category_description	varchar	text	deskripsi singkat kategori
category_banner	varchar	300	baner untuk kategori
category_published	integer	11	keterangan ditampilkan (ya atau tidak)

Tabel *cart_item* memiliki 5 *field* dengan *cart_item_id* sebagai *primary key*. Detail tabel *cart_item* secara lengkap disajikan pada tabel 3.6.

Tabel 3.6. Desain tabel *cart_item*

Nama Kolom	Tipe Data	Lebar	Keterangan
cart_item_id	integer	11	*primary key, auto_increment*
cart_id	varchar	300	*foreign key*
product_id	integer	11	*foreign key* produk
cart_item_qty	integer	11	jumlah barang
total_price	integer	11	total yang dihasilkan

Tabel *shopping_cart* memiliki 2 field dengan *cart_id* sebagai *primary key*. Detail tabel *shopping_cart* secara lengkap disajikan pada table 3.7.

Tabel 3.7. Desain tabel *shopping_cart*

Nama Kolom	Tipe Data	Lebar	Keterangan
cart_id	varchar	300	*primary key*, dengan isi *session cart*
cart_total_price	Int	11	total harga

Tabel *product_order* memiliki 4 field dengan *order_id* sebagai *primary key*. Sementara yang menjadi *foreign key* adalah *customer_id*, *address_id*, dan *credit_card_id*. Detail tabel *cart* secara lengkap disajikan pada tabel 3.8.

Tabel 3.8. Desain tabel *product_order*

Nama Kolom	Tipe Data	Lebar	Keterangan
order_id	integer	11	*primary key*
customer_id	integer	11	*foreign key* tabel *customer*
address_id	integer	11	*foreign key* tabel *shipping_address*
credit_card_id	integer	11	*foreign key* tabel *payment*
session_code	varchar	300	kode session pemesanan
grand_total	varchar	300	total biaya pemesanan
order_date	varchar	100	tanggal pemesanan
status	varchar	100	status pemesanan

Tabel *detail_order* memiliki 5 *field* dengan *detail_id* sebagai *primary key* dan *order_id, product_id, detail_qty* sebagai *foreign key*. Detail tabel *detail_order* secara lengkap disajikan pada tabel 3.9.

Tabel 3.9. Desain tabel *detail_order*

Nama Kolom	Tipe Data	Lebar	Keterangan
detail_id	Integer	11	*primary key*
order_id	Integer	11	*foreign key* tabel *product_order*

product_id	Integer	11	*foreign key* tabel *product*
detail_qty	Integer	11	jumlah barang
total_price	Integer	255	total harga barang

Tabel *customer* memiliki 7 *field* dengan *customer_id* sebagai *primary key*. Detail tabel *customer* secara lengkap disajikan pada tabel 3.10.

Tabel 3.10. Desain tabel *customer*

Nama Kolom	Tipe Data	Lebar	Keterangan
customer_id	integer	11	*primary key*
customer_name	varchar	300	nama pelanggan
customer_phone	varchar	255	nomor telefon pelanggan
customer_email	varchar	255	email pelanggan
customer_gender	varchar	30	jenis kelamin pelanggan
customer_birthday	varchar	20	tanggal lahir pelanggan
customer_birthmonth	varchar	20	bulan lahir pelanggan
customer_birttyear	varchar	20	tahun lahir pelanggan
customer_password	varchar	100	password pelanggan

Tabel *credit_card* memiliki 4 *field* dengan *credit_card_id* sebagai *primary key*. Detail tabel *credit_card* secara lengkap disajikan pada tabel 3.11.

Tabel 3.11. Desain tabel *credit_card*

Nama Kolom	Tipe Data	Lebar	Keterangan
credit_card_id	integer	11	*primary key*
type	varchar	100	jenis kartu kredit
number	varchar	100	nomor kartu kredit
name	varchar	100	nama pemilik kartu kredit
expired_month	varchar	100	bulan kadaluarsa kartu kredit
expired_year	varchar	10	tahun kadaluarsa kartu kredit
ccv_cvv	varchar	10	nomor validasi ccv/cvv

Tabel *shipping_address* memiliki 8 field dengan address_id sebagai primary key. Detail tabel *shipping_address* secara lengkap disajikan pada tabel 3.12.

Tabel 3.12. Desain tabel *shipping_address*

Nama Kolom	Tipe Data	Lebar	Keterangan
address_id	integer	11	*primary key*
nama_kontak	varchar	300	nama kontak pengiriman
no_hp	varchar	30	nomor handphone
agent	varchar	100	nama agen pengiriman
address_name	varchar	300	alamat lengkap
provinsi	varchar	100	nama provinsi
kota	varchar	100	nama kota
kecamatan	varchar	100	nama kecamatan
biaya_kirim	integer	100	biaya pengiriman

Database *Web Service*

Tabel *payment* memiliki 8 *field* dengan *address_id* sebagai *primary key*. Detail tabel *address* secara lengkap disajikan pada tabel 3.13.

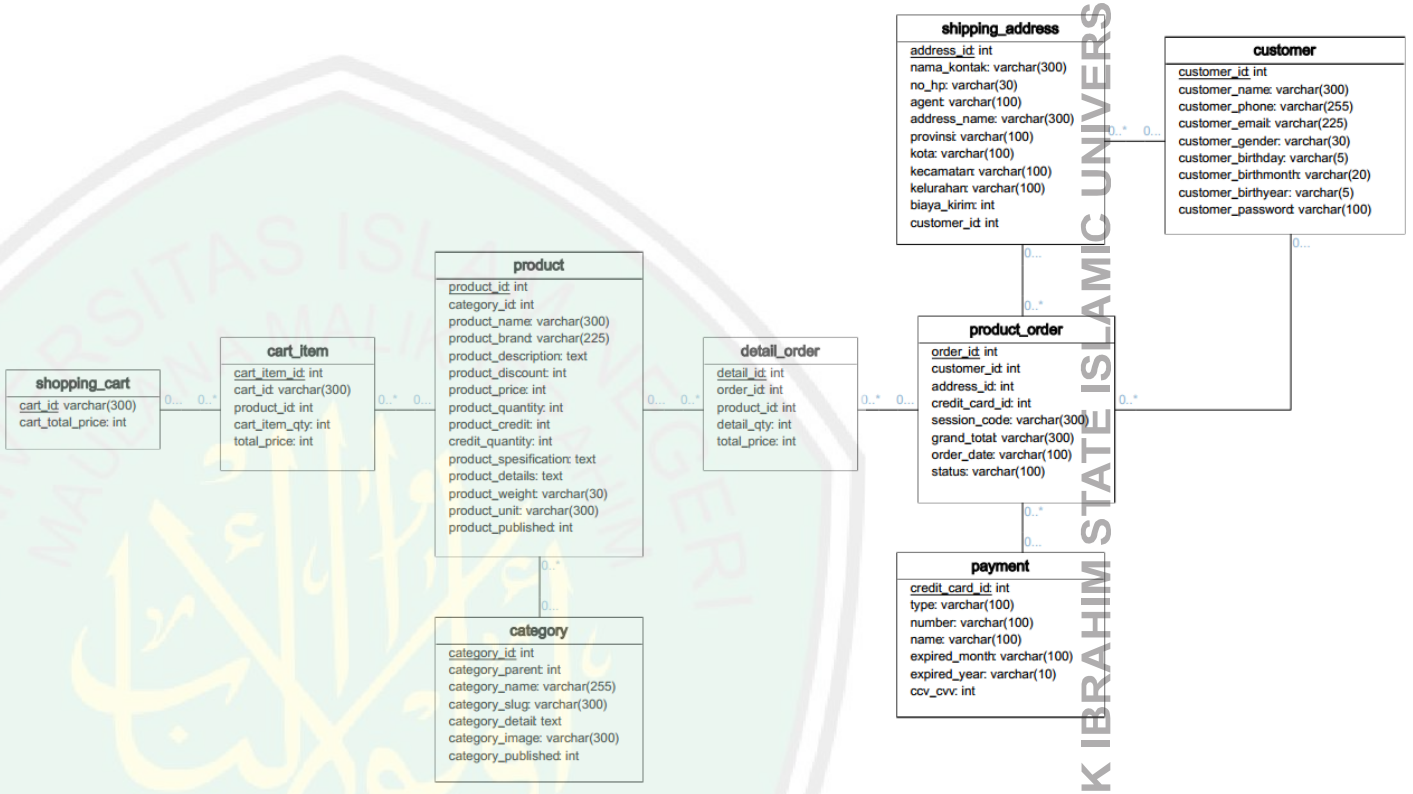
Tabel 3.13. Desain tabel *payment*

Nama Kolom	Tipe Data	Lebar	Keterangan
credit_card_id	integer	11	*primary key*
type	varchar	100	jenis kartu kredit
number	varchar	100	nomor kartu kredit
name	varchar	100	nama pemilik kartu kredit
expired (mm/yy)	varchar	100	bulan tahun kadaluarsa kartu kredit
ccv_cvv	varchar	10	nomor validasi ccv/cvv

Tabel *shipping* memiliki 5 *field* dengan *id_shipping* sebagai *primary key*. Detail tabel *shipping* secara lengkap disajikan pada tabel 3.14.

Tabel 3.14. Desain tabel *shipping*

Nama Kolom	Tipe Data	Lebar	Keterangan
id_shipping	integer	255	*primary key*
shiping_agent	varchar	255	nomor kartu kredit
form	varcha	255	tanggal kadaluarsa
destionation	integer	255	sisa kredit
costs	integer	11	biaya pengiriman



Gambar 3.35. *Entity Relationship Diagram E-Commerce*

BAB IV

IMPLEMENTASI SISTEM

Implementasi merupakan proses yang dilakukan untuk membangun sistem berdasarkan rancangan yang telah dibuat. Dalam hal ini adalah sistem *e-commerce* dengan SOA. Implementasi sistem ini juga tidak hanya tentang kebutuhan perangkat lunak saja, akan tetapi juga kebutuhan perangkat keras. hal ini diperlukan untuk mengukur seberapa mampu *resource* perangkat keras yang ada dalam mendukung perangkat lunak yang akan dibangun. kemudian pada bab ini akan dipaparan implementasi sistem, *back end web service* beserta tampilan sistem yang telah dibangun.

4.1. Kebutuhan Perangkat Keras

Dalam proses pengembangan sistem *e-commerce* ini menggunakan komputer dengan spesifikasi perangkat keras sebagai berikut:

1. CPU Processor Intel® Core™ i3-2370M 2.4GHz
2. HDD Storage 500GB
3. RAM Memory 4 GB
4. Graphic Card NVIDIA GEFORCE 610M 2GB

4.2. Kebutuhan Perangkat Lunak

Dalam proses pengembangan Sistem *E-Commerce* ini menggunakan computer dengan spesifikasi perangkat lunak sebagai berikut :

1. Sistem Operasi Windows 8.1 64 Bit
2. Bahasa Pemrograman Java dan XML

3. IDE Netbeans 8.0 dan Spring Tool Suite
4. Apache Maven 3.2.5
5. Metro Web Service
6. Apache Tomcat 8.0.2
7. Spring Framework 4
8. StarUML 5
9. Apache Tiles

4.3. Implementasi Sistem

4.3.1. Implementasi Basis Data

Sistem *e-commerce* ini diimplemetasikan dalam sebuah basis data dengan nama *db_ecommerce* pada *localhost*. Ada 9 tabel dalam basis data *db_ecommerce* yaitu tabel *product, category, customer, order, shopping cart, cart item, payment* dan *shipping*. Masing-masing diimplementasikan dengan SQL dalam DBMS MySQL. Berikut merupakan *script* yang digunakan untuk membangun tabel dan database seperti yang dijelaskan diatas:

```
-- ----------------------------
-- Table structure for cart_item
-- ----------------------------
DROP TABLE IF EXISTS `cart_item`;
CREATE TABLE `cart_item` (
  `cart_item_id` int(11) NOT NULL AUTO_INCREMENT,
  `cart_id` varchar(300) DEFAULT NULL,
  `product_id` int(11) DEFAULT NULL,
  `cart_item_qty` int(11) DEFAULT NULL,
  `total_price` int(11) DEFAULT NULL,
  PRIMARY KEY (`cart_item_id`),
  KEY `fk_rel_cart_item_cart` (`cart_id`),
```

```
  KEY `fk_rel_cart_item_product` (`product_id`),
  CONSTRAINT `fk_rel_cart_item_cart` FOREIGN KEY (`cart_id`)
REFERENCES `shopping_cart` (`cart_id`),
  CONSTRAINT `fk_rel_cart_item_product` FOREIGN KEY
(`product_id`) REFERENCES `product` (`product_id`)
) ENGINE=InnoDB AUTO_INCREMENT=37 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for category
-- ----------------------------
DROP TABLE IF EXISTS `category`;
CREATE TABLE `category` (
  `category_id` int(11) NOT NULL AUTO_INCREMENT,
  `category_parent` int(11) DEFAULT NULL,
  `category_name` varchar(255) DEFAULT NULL,
  `category_slug` varchar(300) DEFAULT NULL,
  `category_detail` text,
  `category_image` varchar(300) DEFAULT NULL,
  `category_published` int(11) DEFAULT NULL,
  PRIMARY KEY (`category_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1002 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for customer
-- ----------------------------
DROP TABLE IF EXISTS `customer`;
CREATE TABLE `customer` (
  `customer_id` int(11) NOT NULL AUTO_INCREMENT,
  `customer_name` varchar(300) DEFAULT NULL,
  `customer_phone` varchar(255) DEFAULT NULL,
  `customer_email` varchar(225) DEFAULT NULL,
  `customer_gender` varchar(30) DEFAULT NULL,
  `customer_birthday` varchar(5) DEFAULT NULL,
  `customer_birthmonth` varchar(20) DEFAULT NULL,
  `customer_birthyear` varchar(5) DEFAULT NULL,
  `customer_password` varchar(100) DEFAULT NULL,
```

```
  PRIMARY KEY (`customer_id`)
) ENGINE=InnoDB AUTO_INCREMENT=36 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for detail_order
-- ----------------------------
DROP TABLE IF EXISTS `detail_order`;
CREATE TABLE `detail_order` (
  `detail_id` int(11) NOT NULL AUTO_INCREMENT,
  `order_id` int(11) DEFAULT NULL,
  `product_id` int(11) DEFAULT NULL,
  `detail_qty` int(11) DEFAULT NULL,
  `total_price` int(255) DEFAULT NULL,
  PRIMARY KEY (`detail_id`),
  KEY `fk_rel_order_detail` (`order_id`),
  KEY `fk_rel_product_detail` (`product_id`),
  CONSTRAINT `fk_rel_order_detail` FOREIGN KEY (`order_id`)
REFERENCES `product_order` (`order_id`),
  CONSTRAINT `fk_rel_product_detail` FOREIGN KEY (`product_id`)
REFERENCES `product` (`product_id`)
) ENGINE=InnoDB AUTO_INCREMENT=76 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for payment
-- ----------------------------
DROP TABLE IF EXISTS `payment`;
CREATE TABLE `payment` (
  `credit_card_id` int(11) NOT NULL AUTO_INCREMENT,
  `type` varchar(100) DEFAULT NULL,
  `number` varchar(100) DEFAULT NULL,
  `name` varchar(100) DEFAULT NULL,
  `expired_month` varchar(100) DEFAULT NULL,
  `expired_year` varchar(10) DEFAULT NULL,
  `ccv_cvv` int(10) DEFAULT NULL,
  PRIMARY KEY (`credit_card_id`)
) ENGINE=InnoDB AUTO_INCREMENT=100 DEFAULT CHARSET=latin1;
```

```
-- ----------------------------
-- Table structure for product
-- ----------------------------
DROP TABLE IF EXISTS `product`;
CREATE TABLE `product` (
  `product_id` int(11) NOT NULL AUTO_INCREMENT,
  `category_id` int(11) DEFAULT NULL,
  `product_name` varchar(300) DEFAULT NULL,
  `product_brand` varchar(225) DEFAULT NULL,
  `product_description` text,
  `product_discount` int(11) DEFAULT NULL,
  `product_price` int(11) DEFAULT NULL,
  `product_quantity` int(11) DEFAULT NULL,
  `product_credit` int(11) DEFAULT NULL,
  `credit_quantity` int(11) DEFAULT NULL,
  `product_spesification` text,
  `product_details` text,
  `product_weight` varchar(30) DEFAULT NULL,
  `product_unit` varchar(300) DEFAULT NULL,
  `product_published` int(11) DEFAULT NULL,
  PRIMARY KEY (`product_id`),
  KEY `fk_rel_product_category` (`category_id`),
  CONSTRAINT `fk_rel_product_category` FOREIGN KEY
(`category_id`) REFERENCES `category` (`category_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2106 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for product_order
-- ----------------------------
DROP TABLE IF EXISTS `product_order`;
CREATE TABLE `product_order` (
  `order_id` int(11) NOT NULL AUTO_INCREMENT,
  `customer_id` int(11) DEFAULT NULL,
  `address_id` int(11) DEFAULT NULL,
  `credit_card_id` int(11) DEFAULT NULL,
```

```
  `session_code` varchar(300) DEFAULT NULL,
  `grand_total` varchar(300) DEFAULT NULL,
  `order_date` varchar(100) DEFAULT NULL,
  `status` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`order_id`),
  KEY `fk_relationship_9` (`address_id`),
  KEY `fk_rel_customer_order` (`customer_id`),
  KEY `fk_rel_order_payment` (`credit_card_id`),
  CONSTRAINT `fk_relationship_9` FOREIGN KEY (`address_id`)
REFERENCES `shipping_address` (`address_id`),
  CONSTRAINT `fk_rel_customer_order` FOREIGN KEY (`customer_id`)
REFERENCES `customer` (`customer_id`),
  CONSTRAINT `fk_rel_order_payment` FOREIGN KEY
(`credit_card_id`) REFERENCES `payment` (`credit_card_id`)
) ENGINE=InnoDB AUTO_INCREMENT=112 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for shipping_address
-- ----------------------------
DROP TABLE IF EXISTS `shipping_address`;
CREATE TABLE `shipping_address` (
  `address_id` int(11) NOT NULL AUTO_INCREMENT,
  `nama_kontak` varchar(300) DEFAULT NULL,
  `no_hp` varchar(30) DEFAULT NULL,
  `agent` varchar(100) DEFAULT NULL,
  `address_name` varchar(300) DEFAULT NULL,
  `provinsi` varchar(100) DEFAULT NULL,
  `kota` varchar(100) DEFAULT NULL,
  `kecamatan` varchar(100) DEFAULT NULL,
  `kelurahan` varchar(100) DEFAULT NULL,
  `biaya_kirim` int(100) DEFAULT NULL,
  `customer_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`address_id`),
  KEY `fk_customer` (`customer_id`),
  CONSTRAINT `fk_customer` FOREIGN KEY (`customer_id`) REFERENCES
`customer` (`customer_id`)
```

```
) ENGINE=InnoDB AUTO_INCREMENT=123 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for shopping_cart
-- ----------------------------
DROP TABLE IF EXISTS `shopping_cart`;
CREATE TABLE `shopping_cart` (
  `cart_id` varchar(300) NOT NULL,
  `cart_total_price` int(11) DEFAULT NULL,
  PRIMARY KEY (`cart_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Selanjutnya adalah melakukan penerapan konsep *Service Oriented Archtecture (SOA)* yang diimplementasikan kedalam *web service.* Peneliti menggunakan 3 permodelan basis data yang mewakili struktur database pada web service asli, model yang dibangun mengacu pada penelitian (Baraka & Al Aqshar, 2013) yang berjudul *"Building a SOA-Based Model for Purchase Order Management in E-Commerce Systems".*

Basis data pertama adalah basis data *db_payment.* Basis data ini berfungsi mewakili data yang ada dalam *bank.* Tujuannya adalah untuk melakukan pengecekan credit pada kartu kredit. Metode pembayaran pada aplikasi ini dibatasi pada pembayaran kartu kredit. Berikut merupakan *script* implementasi *db_payment* pada database MySQL.

```
-- ----------------------------
-- Table structure for master_card
-- ----------------------------
DROP TABLE IF EXISTS `master_card`;
CREATE TABLE `master_card` (
  `id_card` int(255) NOT NULL AUTO_INCREMENT,
  `number` varchar(255) DEFAULT NULL,
```

```
  `name` varchar(255) DEFAULT NULL,
  `expired` varchar(255) DEFAULT NULL,
  `credits` int(255) DEFAULT NULL,
  `ccv_cvv` int(10) DEFAULT NULL,
  PRIMARY KEY (`id_card`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;
```

Basis data yang kedua adalah basis data *db_shipping*. Basis data ini berfungsi mewakili data yang ada dalam agen pengiriman barang. Tujuannya adalah untuk melakukan pengecekan biaya kirim sesuai dengan jumlah barang dan alamat tujuan. Berikut merupakan *script* implementasi dalam SQL pada database MySQL.

```
-- ----------------------------
-- Table structure for shipping
-- ----------------------------
DROP TABLE IF EXISTS `shipping`;
CREATE TABLE `shipping` (
  `id_shipping` int(255) NOT NULL AUTO_INCREMENT,
  `agent` varchar(255) DEFAULT NULL,
  `from` varchar(255) DEFAULT NULL,
  `destination` varchar(255) DEFAULT NULL,
  `costs` int(255) DEFAULT NULL,
  PRIMARY KEY (`id_shipping`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;
```

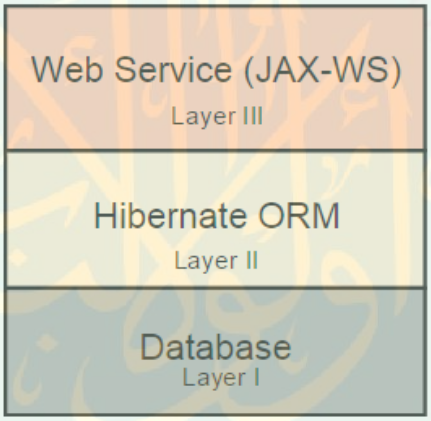
Basis data yang ketiga adalah basis data *db_inventory*. Basis data ini berfungsi mewakili data yang ada dalam Sistem *Inventory*. Tujuannya adalah untuk melakukan simulasi pengecekan ketersediaan barag pada gudang. Berikut merupakan *script* implementasi dalam SQL pada database MySQL.

```
-- ----------------------------
-- Table structure for item
-- ----------------------------
DROP TABLE IF EXISTS `item`;
CREATE TABLE `item` (
  `kode_item` varchar(255) NOT NULL,
  `id_suppliers` varchar(300) DEFAULT NULL,
  `id_cat_barang` int(11) DEFAULT NULL,
  `id_satuan` int(11) DEFAULT NULL,
  `nama_item` varchar(300) DEFAULT NULL,
  `brand` varchar(225) DEFAULT NULL,
  `detail_tambahan` text,
  `diskon` int(11) DEFAULT NULL,
  `harga_beli` int(11) DEFAULT NULL,
  `qty_cicilan` int(11) DEFAULT NULL,
  `besar_cicilan` int(11) DEFAULT NULL,
  `harga_jual` int(11) DEFAULT NULL,
  `spesifikasi` text,
  `detail_produk` text,
  `berat` int(11) DEFAULT NULL,
  `qty_item` int(11) DEFAULT NULL,
  `published` int(11) DEFAULT NULL,
  PRIMARY KEY (`kode_item`),
  KEY `fk_rel_product_category` (`id_cat_barang`),
  KEY `fk_rel_suppliers_product` (`id_suppliers`),
  KEY `fk_satuan` (`id_satuan`),
  CONSTRAINT `fk_rel_product_category` FOREIGN KEY
(`id_cat_barang`) REFERENCES `category` (`id_cat_barang`),
  CONSTRAINT `fk_rel_suppliers_product` FOREIGN KEY
(`id_suppliers`) REFERENCES `suppliers` (`id_suppliers`),
  CONSTRAINT `fk_satuan` FOREIGN KEY (`id_satuan`)
REFERENCES `satuan` (`id_satuan`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

4.3.2. Implementasi *Web Service*

Sistem *e-commerce* menggunakan *web service* untuk mengimplementasikan SOA. Dalam pembangunan *web service* ini peneliti mengimplementasikannya dalam bahasa Java yang diakses melalui teknologi *Object Relational Mapping (ORM)* setelah itu diproses sebagai *web service* dengan JAX-WS Metro. Struktur implemetasi *web service* terlihat pada gambar 4.1.



Gambar 4.1. *Model layer web service*

Layer I menjelaskan tentang struktur basis data yang akan dijadikan sumber. *Layer* II merupakan struktur pengaksesan basis data dengan model ORM, sedangkan *Layer* II merupakan layer pengeksposan data dengan *web service.* Basis data yang digunakan pada *layer* I adalah MySQL. *Web service* yang dibangun pada *layer* III menggunakan *Metro* yang merupakan *library* implementasi dari JAX-WS. Detail *web service* yang digunakan beserta operasinya akan dijelaskan dalam tabel sebagai berikut.

Tabel 4.1. Operasi *web services*

NO	Nama Operasi	Parameter Input	Output
1	validateCreditCard()	String CCNumber, String ExpDate, String Amount	Status *valid/invalid*
2	setCreditAmount()	String CCNumber, Name, ExpiredMonth, ExpiredYear, CCV/CVV, Amount	Status *success*
3	getShippingCost()	String agent, String address	*Shipping costs*
4	checkItem()	String IdItem, String Qty	Status *instock/outstock*
5	SetQtyItem()	int Qty, String idItem	Status *success*
6	calculatePayment()	Int Qty, int price, int shippingcosts	Total bill/invoice

Penjelasan secara detail mengenai operasi dalam web service pada tabel 4.1 adalah sebagai berikut:

1. validateCreditCard(*parameter input*), merupakan fungsi untuk mengecek status aktif dan sisa kredit dari kartu kredit. Dengan cara menginputkan parameter nama bank, *expired date* dan jumlah kredit. Apabila kartu kredit sudah tidak aktif masa berlakunya maka akan memberikan pesan *invalid.* Namun, apabila kartu kredit masih aktif maka *valid* dan proses berlanjut pada pengecekan kredit. Apabila kredit lebih besar atau sama dengan jumlah total harga maka kartu kredit tersebut baru dinyatakan *valid*. Berikut fungsi *validateCreditCard()* terlihat pada gambar 4.2.

```
@WebService(serviceName = "MasterCardService")
public class MasterCardService {
    @WebMethod(operationName = "validateCreditCard")
    public String validateCreditCard(
            @WebParam(name = "CCNumber") String CCNumber,
            @WebParam(name = "Name") String Name,
            @WebParam(name = "ExpMonth") String ExpMonth,
            @WebParam(name = "ExpYear") String ExpYear,
            @WebParam(name = "CCV_CVV") Integer CCV_CVV,
            @WebParam(name = "Amount") int Amount) {
        Session session = NewHibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        String hql = "FROM MasterCard WHERE number=:Number AND name=:Name AND ccvCvv=:CcvCvv AND "
                + "expired=:Expired";
        Query query = session.createQuery(hql);
        query.setParameter("Number", CCNumber);
        query.setParameter("Name", Name);
        query.setParameter("CcvCvv", CCV_CVV);
        String Expired=ExpMonth+"/"+ExpYear;
        query.setParameter("Expired", Expired);
        List<MasterCard> list = query.list();
        String status = "invalid";
        if(query.list().size()==0){
            return status;
        }
        for (int a = 0; a < list.size(); a++) {
            Date date = new Date();
            MasterCard masterCard = (MasterCard) query.list().get(a);
            if (masterCard.getNumber().equals(CCNumber)) {
                if (masterCard.getExpired().compareTo(Expired) == 0) {
                    try {
```

Gambar 4.2. fungsi *validateCreditCard()*.

2. setCreditAmount(*input parameter*), merupakan fungsi untuk mengurangi jumlah kredit yang berada dalam kartu kredit. Parameter yang digunakan sama dengan *validateCreditCard()*. Apabila total pembelian telah terhitung, maka jumlah kredit yang ada dalam kartu kredit akan dikurangi sebesar total biaya belanja tersebut. Berikut fungsi *setCreditAmount()* terlihat pada gambar 4.3.

```
@WebMethod(operationName = "setCreditAmount")
public String setCreditAmount(
        @WebParam(name = "CCNumber") String CCNumber,
        @WebParam(name = "Amount") int Amount) {
    Session session = NewHibernateUtil.getSessionFactory().openSession();
    Transaction tr = session.beginTransaction();
    Query query = session.createQuery("from CreditCard where ccNumber= :CCNumber");
    query.setParameter("CCNumber", CCNumber);
    List list = query.list();
    for (int a = 0; a < list.size(); a++) {
        CreditCard creditCard = (CreditCard) list.get(a);
        creditCard.setCredits(creditCard.getCredits()-Amount);
    }
    tr.commit();
    return "sukses update";
}
```

Gambar 4.3. fungsi *setCreditAmount ().*

3. getShippingCost(*parameter input*), merupakan fungsi untuk pengecekan biaya pengiriman. Parameter yang digunakan adalah *shipping agent* dan kota tujuan. Apabila pengecekan sukses maka akan memberikan output biaya pengiriman. Berikut fungsi *getShippingCost()* terlihat pada gambar 4.4.

```
@WebService(serviceName = "ShippingService")
public class ShippingService {
    Session session = NewHibernateUtil.getSessionFactory().openSession();
    @WebMethod(operationName = "RequireShipping")
    public int getShippingCost(@WebParam(name = "Agent") String Agent,
            @WebParam(name = "Address") String Address) {
        Session session = NewHibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        Integer costs = 0;
        String hql = "FROM Shipping WHERE agent=:Agent AND destination=:Address";
        Query query = session.createQuery(hql);
        query.setParameter("Agent", Agent);
        query.setParameter("Address", Address);
        List<Shipping> list = query.list();
        for (int a = 0; a < list.size(); a++) {
            Shipping shipping = (Shipping) query.list().get(a);
            if (Agent.compareTo(shipping.getAgent()) == 0) {
                costs = shipping.getCosts();
            }
        }
        session.close();
        return costs;
    }
}
```

Gambar 4.4. fungsi *getShippingCost ().*

4. checkItem(*parameter input*), merupakan fungsi untuk pengecekan ketersedian pada gudang. Parameter yang digunakan adalah SKU *(Stock Keeping Unit)* dan kuantitas barang. apabila barang tersedia maka dapat dipesan oleh pembeli. Sebaliknya, bila barang tidak tersedia maka akan menampilkan pesan *out stock* dn secara otomatis pembeli tidak dapat memesannya. Berikut implementasi fungsi *checkItem()* terlihat pada gambar 4.5.

```
@WebService(serviceName = "InventoryService")
public class InventoryService {
    @WebMethod(operationName = "CheckItem")
    public String CheckItem(@WebParam(name = "IdItem") String IdItem,
            @WebParam(name = "Qty") int Qty) {
        Session session = NewHibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        Query query = session.createQuery("FROM Item WHERE kodeItem=:id");
        query.setParameter("id", IdItem);
        List<Item> ls = query.list();
        for (int i = 0; i < ls.size(); i++) {
            Item item = (Item) query.list().get(i);
            if(String.valueOf(item.getKodeItem()).compareTo(IdItem)==0){
                if(Qty<=item.getQtyItem()){
                    session.close();
                    return "valid";
                }else{
                    return "invalid";
                }
            }
        }
        session.close();
        return "invalid";
    }
}
```

Gambar 4.5. fungsi *validateItem()*

5. setQtyItem(*parameter input*), merupakan fungsi untuk mengubah kuantitas barang yang dipesan pada gudang. Parameter yang digunakan adalah kode SKU dan kuantitas jumlah barang yang dipesan. Berikut implementasi *listing code setQtyItem()* terlihat pada gambar 4.6.

```
@WebMethod(operationName = "SetQtyItem")
public String setQtyItem(@WebParam(name = "Qty") int Qty,
        @WebParam(name = "idItem") String idItem) {
    Session session = NewHibernateUtil.getSessionFactory().openSession();
    Transaction tr = session.beginTransaction();
    Query query = session.createQuery("from Item where kodeItem= :id");
    query.setParameter("id", idItem);
    List list = query.list();
    for (int a = 0; a < list.size(); a++) {
        Item item = (Item) list.get(a);
        Item i = (Item) session.load(Item.class, idItem);
        i.setQtyItem(item.getQtyItem()- Qty);
    }
    tr.commit();
    return "sukses update";
}
```

Gambar 4.6. fungsi *setQtyItem()*

6. calculatePayment(*parameter input*), merupakan fungsi untuk menghitung total tagihan *order*. Parameter yang digunakan adalah jumlah barang, harga dan biaya kirim. Berikut implementasi *listing code setQtyItem()* terlihat pada gambar 4.6.

```
@WebMethod(operationName = "CalculatePayment")
public int CalculatePayment(@WebParam(name = "Qty") int Qty,
        @WebParam(name = "Price") int Price,
        @WebParam(name = "ShippingCosts") int ShipingCosts) {
    return Price * Qty + ShipingCosts;
}
```

Gambar 4.7. fungsi *calculatePayment()*

4.3.3. Implementasi Sistem

Sistem *e-commerce* dengan SOA ini diimplementasikan dengan bahasa pemrograman java dengan menggunakan *framework* Spring. Spring merupakan salah satu standar implementasi *JEE (Java Enterprise Edition)*. Java dan Spring telah menyediakan komponen pembangunan untuk aplikasi *enterprise* dan terintegrasi seperti SOA. Sebelum menggunakan Spring, maka kita harus mengatur konfigurasi basis data yang digunakan dengan setingan pada *DispatcherServlet-context.xml*. Berikut ini *listing code* untuk melakukan konfigurasi *database*, terlihat pada gambar 4.7.

```
<tx:annotation-driven transaction-manager="transactionManager" />
<bean class="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" id="dataSource">
    <property name="user" value="root" />
    <property name="password" value="" />
    <property name="databaseName" value="db_webstore" />
    <property name="serverName" value="localhost" />
    <property name="portNumber" value="3306" />
</bean>
<bean class="org.springframework.orm.hibernate4.HibernateTransactionManager" id="transactionManager">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
<bean class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" id="sessionFactory">
    <property name="packagesToScan" value="com.webstore.model" />
    <property name="dataSource" ref="dataSource" />
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
            <prop key="hibernate.enable_lazy_load_no_trans">true</prop>
        </props>
    </property>
</bean>
```

Gambar 4.8. *listing code* koneksi *database* Spring dengan Hibernate ORM

4.3.3.1. Implementasi *Login*

Halaman login adalah halaman yang akan mucul ketika *customer* melakukan *checkout* atau admin masuk halaman *administrator*. Proses login berfungsi memfilter hak akses dalam menggunakan suatu sistem. *Customer* memiliki hak akses yang berbeda dengan *administrator* begitu pula sebaliknya. Proses *login* ini terdapat 2 tipe, pertama *login* untuk *customer* dan yang kedua adalah *administrator*. Apabila *customer* telah selesai melakukan pemilihan barang dan ingin melanjukan ke proses *checkout* maka harus *login* terlebih dahulu. Apabila *administrator* ingin melakukan manajemen situs secara utuh maka harus *login* terlebih dahulu. Pada saat melakukan *login*, maka akan ditampilkan form *login* dari *view* yang dipanggil melalui *ManagerController*. Berikut merupakan *listing code* pada *security-conetxt.xml* dan *ManagerController* untuk *login* sebagai *admin* dapat dilihat pada gambar 4.8 dan 4.9.

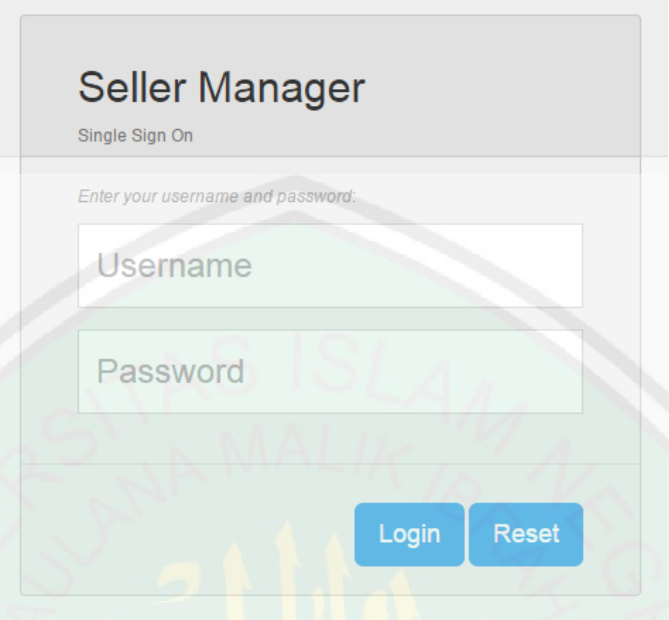
```
<security:http auto-config="true">
    <security:intercept-url pattern="/manager/product*" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/category**" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/order*" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/customer*" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/product/**" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/category/**" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/order/**" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/customer/**" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/manager/index" access="ROLE_ADMIN" />
    <security:form-login login-page="/manager/login"
                         default-target-url="/manager/"
                         authentication-failure-url="/manager/loginfailed"/>
    <security:logout logout-success-url="/manager/logout" />
</security:http>

<security:authentication-manager>
    <security:authentication-provider>
        <security:user-service>
            <security:user name="admin" password="admin123" authorities="ROLE_ADMIN" />
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>
```

Gambar 4.9. *listing code* konfigurasi security-context.xml

```
//---------Login Manager Start-----------------//
@RequestMapping(value = "/login", method = RequestMethod.GET)
public String login() {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken)) {
        return "redirect:/manager/index";
    }
    return "ManagerLogin";
}

@RequestMapping(value = "loginfailed", method = RequestMethod.GET)
public String loginerror(Model model) {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken)) {
        return "redirect:/manager/index";
    }
    model.addAttribute("error", "true");
    return "ManagerLogin";
}

@RequestMapping(value = "/logout", method = RequestMethod.GET)
public String logout(Model model) {
    model.addAttribute("message", "anda telah logout");
    return "ManagerLogin";
}
//----------Login Manager End--------------------//
```

Gambar 4.10. *listing code* konfigurasi *ManagerController* fungsi *login*

Adapun tampilan halaman *login admin* diperlihatkan pada gambar 4.10. berikut ini.



Gambar 4.11. Tampilan halaman *login administrator*

4.3.3.2. Implementasi Melihat Barang

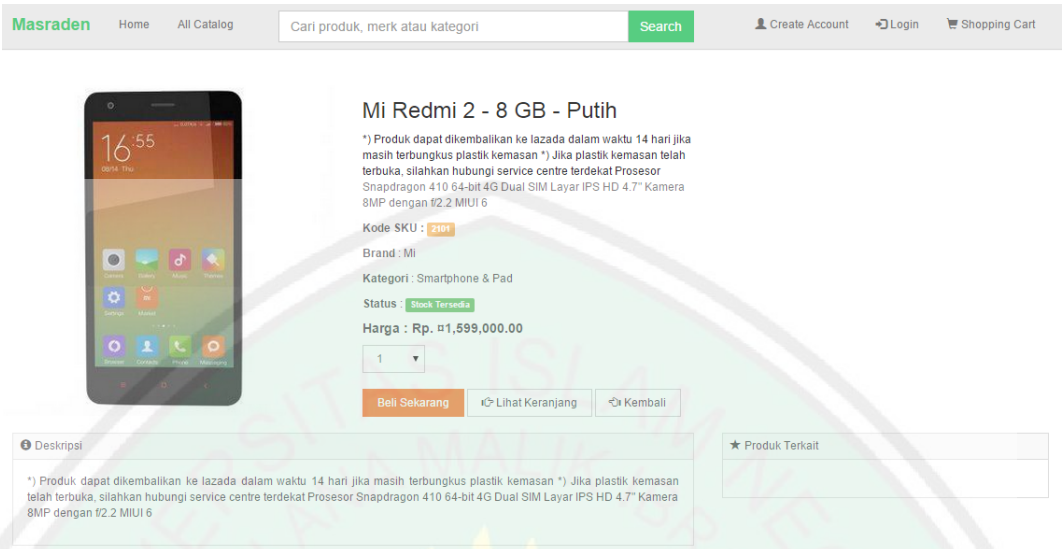
Proses menampilkan daftar barang yang dijual berada pada halaman katalog. Pada halaman ini *customer* dapat memilih barang berdasarkan kategori yang ada. *Controller* yang digunakan adalah *ProductController*, dalam controller ini terdapat 3 fungsi yang pertama *getAllProduct()* yang berfungsi menampilkan semua produk, yang kedua *getProductByCategory()* yang menampilkan produk berdasarkan kategori tertentu dan yang ketiga adalah *getProductById()* yang berfungsi menampilkan deskripsi lengkap sebuah produk berdasarkan id produk tersebut. Berikut *listing code* sebagaimana diperlihatkan pada gambar 4.11.

```
@RequestMapping("/catalog/all")
public String list(Model model) {
    model.addAttribute("products", productService.getAllProducts());
    model.addAttribute("categorys", productService.getAllCategory());
    return "katalog";
}

@RequestMapping("/category/{category}")
public String getProductByCategory(Model model, @PathVariable("category") String category) {
    List<Product> products = productService.getProductByCategory(category);
    if (products == null || products.isEmpty()) {
        throw new NoProductsFoundUnderCategoryException();
    }
    model.addAttribute("categorys", productService.getAllCategory());
    model.addAttribute("products", products);
    return "katalog";
}
@RequestMapping("/item/{id}")
public String getProductById(@PathVariable("id") String productId, Model model) {
    Product products = productService.getProductById(productId);
    if (products.getProductId() == null || products.getProductId().isEmpty()) {
        throw new NoProductsFoundUnderCategoryException();
    }
    model.addAttribute("product", products);
    model.addAttribute("cart_item", new CartItem());
    return "Item";
}
```

Gambar 4.12. *listing code ProductController*

Adapun tampilan terkait halaman katalog yang berhubungan dengan fungsi diatas dapat dilihat pada gambar 4.12 dan 4.13 berikut ini.

Gambar 4.13. Tampilan halaman katalog fungsi *getAllProduct()*



Gambar 4.14. Tampilan halaman katalog *getProductById()*

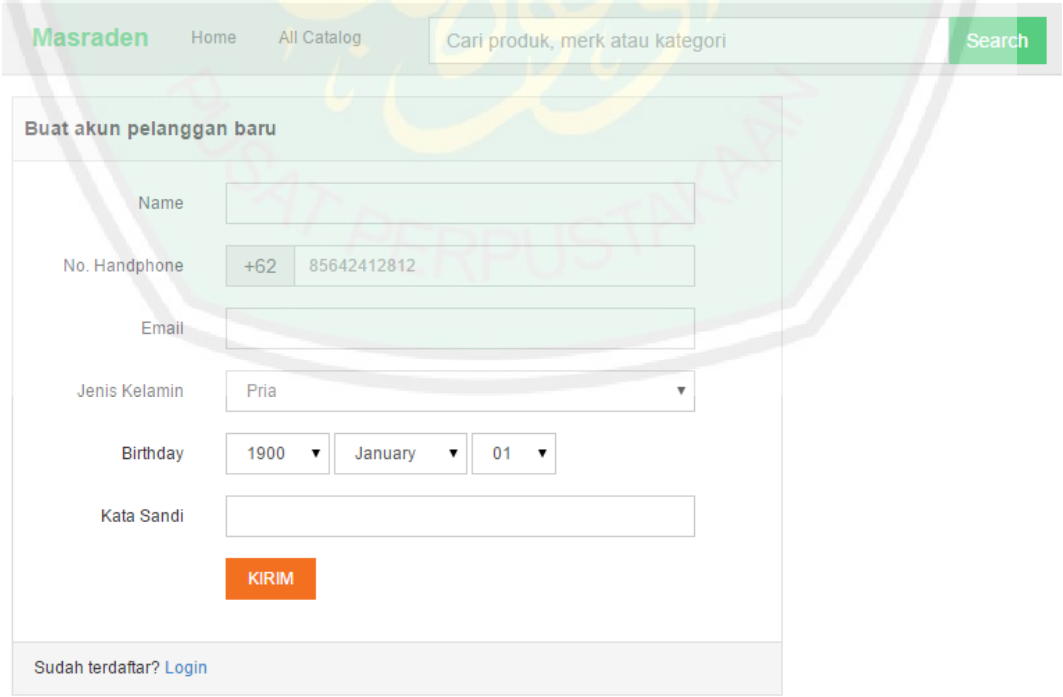
4.3.3.3. Implementasi Registrasi

Proses untuk menampilkan halaman registrasi terletak pada CustomerController. Customer akan mengakses *path url customer/create* yang berada dalam menu *navigation bar create account*. Ketika diklik maka akan memanggil fungsi create pada controller. Dan selanjunya request diterima oleh controller dan ditampilkan ke view dalam bentuk form registrasi. Implementasi *listing code* registrasi customer dapat dilihat seperti gambar 4.14 berikut ini.

```java
@Controller
@RequestMapping(value = "/customer")
public class CustomerController {

    @Autowired
    private CustomerService customerService;

    //for register customer
    @RequestMapping(value = "/create", method = RequestMethod.GET)
    public ModelAndView create(Model model) {
        ModelAndView mav = new ModelAndView("CustomerRegister");
        mav.addObject("customer", new Customer());
        return mav;
    }

    @RequestMapping(value = "/create", method = RequestMethod.POST)
    public String createCustomer(Model model,
            @ModelAttribute("customer") @Valid Customer customer,
            BindingResult result,
            RedirectAttributes attributes) {
        if (result.hasErrors()) {
            return "CustomerRegister";
        }
        customerService.addCustomer(customer);
        String message = "Hi, " + customer.getCustomerName() + ". You are successfully register!";
        attributes.addFlashAttribute("message", message);
        return "redirect:/customer/login";
    }
```

Gambar 4.15. *listing code* registrasi

Adapun tampilan terkait halaman registrasi yang berhubungan dengan fungsi diatas dapat dilihat pada gambar 4.15. berikut ini.



Gambar 4.16. Tampilan halaman registrasi

4.3.3.4. Implementasi Menambahkan ke Keranjang (*add to cart*)

Setelah melihat detail barang, pelanggan dapat menambahkan barang yang ingin dia beli kedalam keranjang. Keranjang belanja dapa diakses dengan *path url cart/* yang nantinya akan mengakses *CartController*. Didalam class *CartControlller* terdapat fungsi *addItem(), deleteItem(), updateItem(), removeAllItem()*.Berikut implementasi *listing code checkout* dan SOA pada *CartController* terlihat pada gambar 4.16.

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
public String addItem(HttpServletRequest request,
        @RequestParam(value = "product_id") String product_id,
        @RequestParam(value = "qty_product") Integer qty_product,
        RedirectAttributes attributes) {
    HttpSession session = request.getSession();
    String sessionId = (String) session.getAttribute("cartId");
    if (sessionId == null || sessionId.equals("")) {
        session.setAttribute("cartId", cartItemService.createId());
    }
    ShoppingCart cart = cartService.read((String) session.getAttribute("cartId"));
    //get cart by session
    ShoppingCart sc = new ShoppingCart();
    Product p = productService.getProductById(product_id);

    if (PomService.CheckItem(product_id, qty_product).equals("valid")) {
        if (cart == null) {
            sc.setCartId((String) session.getAttribute("cartId"));
            cartService.create(sc);
            cartItemService.addItem(sc, p, qty_product);
            return "redirect:/cart";
        } else {
            cartItemService.addItem(cart, p, qty_product);
            return "redirect:/cart";
        }
    } else {
        attributes.addFlashAttribute("message", "This item maybe out of inventory");
        return "redirect:/item/" + product_id;
    }
}
```

Gambar 4.17. *listing code CartController* fungsi *addItem()*

Gambar 4.18. Tampilan keranjang belanja sebelum proses *checkout*

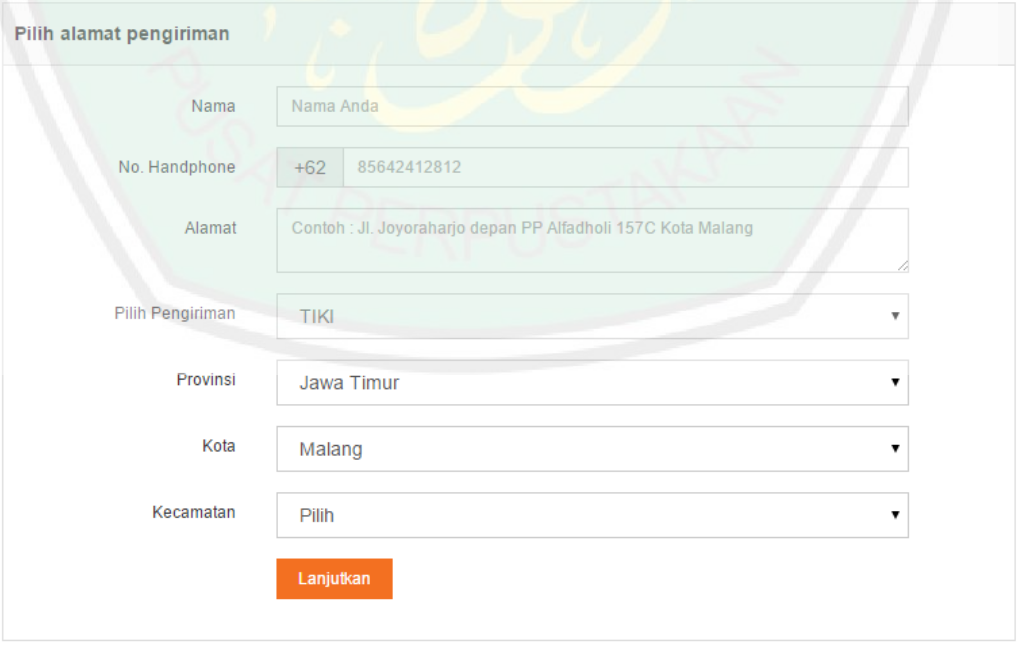
4.3.3.5. Implementasi *Checkout*

Setelah barang yang diinginkan dipilih dan dimasukkan ke keranjang belanja, proses selanjutnya adalah *checkout*. *Checkout* merupakan proses dimana *customer* memasukkan alamat pengiriman serta metode pembayaran, saat *checkout customer* akan mengakses *url* /checkout/order yang nantinya akan memanggil *CheckoutController* dengan fungsi *order()*. Selanjunya akan diteruskan pada fungsi *shipping()*. *Customer* harus mengisi alamat pengiriman, kemudian sistem memvalidasi inputan dan mengkalkulasi biaya pengiriman *(Shipping Service)*. Selanjunya adalah proses pengisian metode pembayaran, dalam hal ini peneliti membatasi pada penggunaan kartu kredit. *Customer* diminta memasukkan kartu kredit dan sistem akan memvalidasi apakah kartu tersebut masih aktif dan apakah total kredit masih mencukupi untuk bertransaksi. Apabila sistem telah memastikan semua data valid maka *checkout* selesai. Dan customer akan menerima *invoice* setelah proses tersebut. Berikut implementasi *listing code checkout* dan SOA pada *CheckoutController* terlihat pada gambar 4.18.

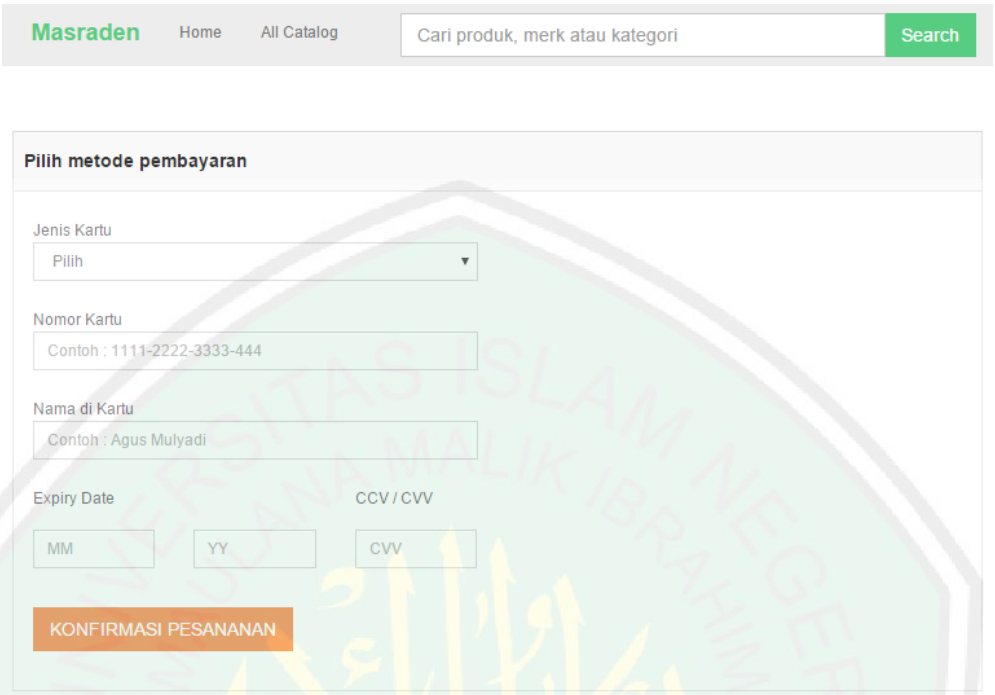
```java
@Controller
@RequestMapping(value = "/checkout")
public class CheckOutController {
    @Autowired
    private OrderService orderService;
    @Autowired
    private POMService pomService;
    @Autowired
    private CartItemService cartItemService;
    @RequestMapping(value = "/order")
    public String order(HttpServletRequest request, RedirectAttributes attributes) {
        HttpSession session = request.getSession();
        String sessionId = (String) session.getAttribute("cartId");
        List<Object[]> item = cartItemService.readItem(sessionId);

        String redirect = "redirect:/checkout/shipping";
        if (item == null || item.isEmpty()) {
            attributes.addFlashAttribute("message", "Keranjang dikosongkan!");
            redirect = "redirect:/cart";
        } else {
            orderService.addOrder(sessionId);
        }
        return redirect;
    }
```

Gambar 4.19. *Listing code CheckoutController* fungsi *order()*

Adapun tampilan terkait halaman katalog yang berhubungan dengan fungsi diatas dapat dilihat pada gambar 4.18 dan 4.19 berikut ini.



Gambar 4.20. Tampilan form *checkout/shipping*



Gambar 4.21. Tampilan form *checkout/payment*

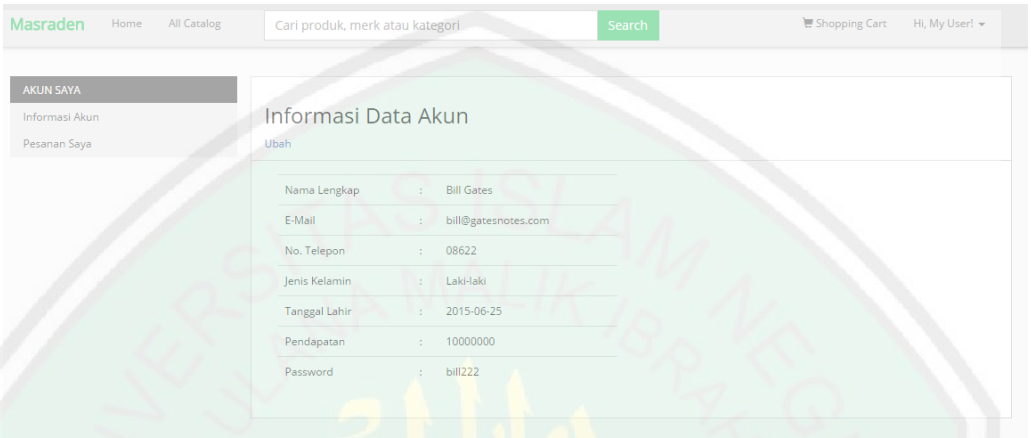
4.3.3.6. Implementasi Mengubah Data Akun (*change account*)

Setelah pelanggan login maka pelanggan bisa melakukan ubah data akun. Hal ini diperlukan jika terjadi perubahan data sewaktu-waktu. Proses ini terletak pada class *CustomerController* dengan fungsi *accountEdit()*. Berikut implementasi *listing code* merubah data akun terlihat pada gambar 4.21.

```
@RequestMapping(value = "/account/edit", method = RequestMethod.GET)
public String accountEdit(Model model,
        @ModelAttribute("customer") @Valid Customer customer,
        BindingResult result,
        RedirectAttributes attributes) {
    if (result.hasErrors()) {
        return "CustomerEdit";
    }
    customerService.editCustomer(customer);
    String message = "Hi, " + customer.getCustomerName() + ". You are successfully edited!";
    attributes.addFlashAttribute("message", message);
    return "redirect:/customer/index";
}
```

Gambar 4.22. *listing code* mengubah data akun

Adapun tampilan halaman merubah data akun sebagaimana dapat dilihat pada gambar 4.22.



Gambar 4.23. Tampilan merubah data akun

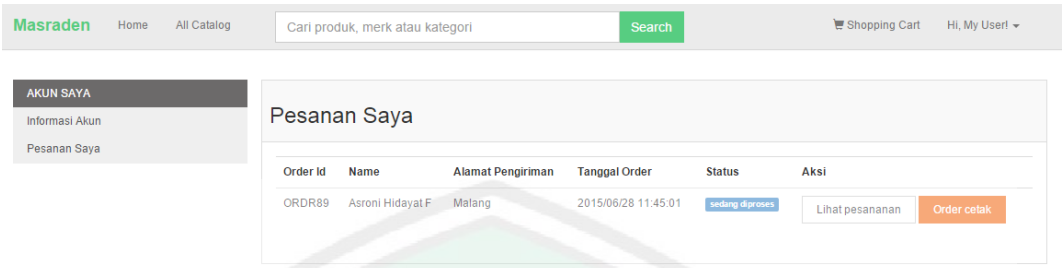
4.3.3.7. Implementasi Melihat Daftar Pemesanan (*order history*)

Setelah pelanggan login maka pelanggan bisa melihat daftar riwayat pemesanan. Proses ini terletak pada class *CustomerController* dengan fungsi *orderByCustomer()*. Berikut implementasi *listing code* melihat daftar pemesanan terlihat pada gambar 4.23.

```
@RequestMapping(value = "/order", method = RequestMethod.GET)
public String orderByCustomer(Model model, HttpServletRequest request) {
    HttpSession session = request.getSession();
    String customerId = (String) session.getAttribute("customerId");
    model.addAttribute("orders", orderService.getOrderByCustomer(customerId));
    return "CustomerOrder";
}
```

Gambar 4.24. *listing code* melihat daftar pemesanan

Adapun tampilan halaman merubah data akun sebagaimana dapat dilihat pada gambar 4.24.



Gambar 4.25. Tampilan melihat daftar pemesanan

4.3.3.8. Implementasi Manajemen Pelanggan (*customer*)

Proses dalam mengelola daftar *customer* yang telah terdaftar adalah pada menu manajemen *customer*. *Administrator* harus *login* terlebih dahulu dengan mengakses *url manager/login*. Setelah *login* sukses *customer* akan di *redirect* pada halaman *admin manager/index*. Salah satu menu yang ada di halaman tersebut adalah manajemen *customer*. Seorang *administrator* diberikan hak akses penuh untuk mengatur data *customer* tersebut, yang meliputi tambah, hapus, ubah dan lihat. Berikut implementasi *listing code* manajemen *customer* pada *ManagerController* terlihat pada gambar 4.25.

```
//---------Start Customer Manager----------------//
@RequestMapping(value = "/customer", method = RequestMethod.GET)
public String customer(Model model) {
    model.addAttribute("customers", customerService.getAllCustomers());
    return "ManagerCustomer";
}

@RequestMapping(value = "/customer/add", method = RequestMethod.GET)
public ModelAndView customerAdd() {
    ModelAndView mav = new ModelAndView("ManagerCustomerAdd");
    mav.addObject("action", "add");
    mav.addObject("customer", new Customer());
    return mav;
}

@RequestMapping(value = "/customer/add", method = RequestMethod.POST)
public String customerAddPost(Model model, @ModelAttribute("customer") @Valid Customer customer,
        BindingResult result, RedirectAttributes attributes) {
    if (result.hasErrors()) {
        model.addAttribute("action", "add");
        return "ManagerCustomerAdd";
    }
    customerService.addCustomer(customer);
    String message = "Hi, " + customer.getCustomerName() + ". You are successfully added!";
    attributes.addFlashAttribute("message", message);
    return "redirect:/manager/customer";
}
```

Gambar 4.26. *listing code* manajemen pelanggan pada *ManagerController*

Adapun tampilan halaman manajemen *customer* sebagaimana dapat dilihat pada gambar 4.26.

Gambar 4.27. Tampilan halaman manajemen pelanggan

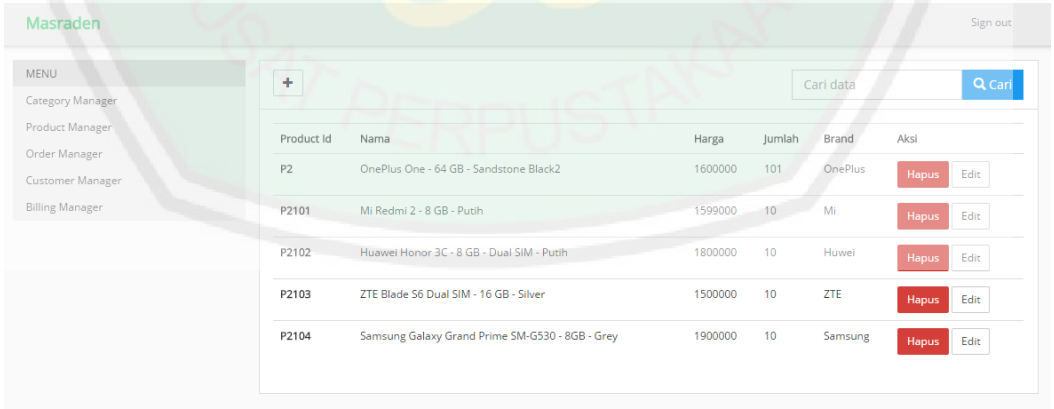
4.3.3.9. Implementasi Manajemen Barang (*product*)

Mengatur barang yang akan dijual adalah hal yang sangat penting dalam sistem *e-commerce*. Untuk itu, dalam menu *administrator* terdapat menu *Product Manager*. Setelah seorang *administrator login*, maka akan memiliki akses untuk melakukan manajemen produk pada *sistem ecommerce* tersebut. *Admin* memiliki hak penuh dalam menambah, menghapus serta mengubah detail informasi produk yang ada. Ketika menu ini diakses maka akan memanggil *path url /manager/product* dan seketika itu mengakses *ManagerController* dengan fungsi *product(). Administrator* dapat menambah produk melalui fungsi *productAdd().* Berikut implementasi *listing code* manajemen *katalog* pada *ManagerController* terlihat pada gambar 4.27.

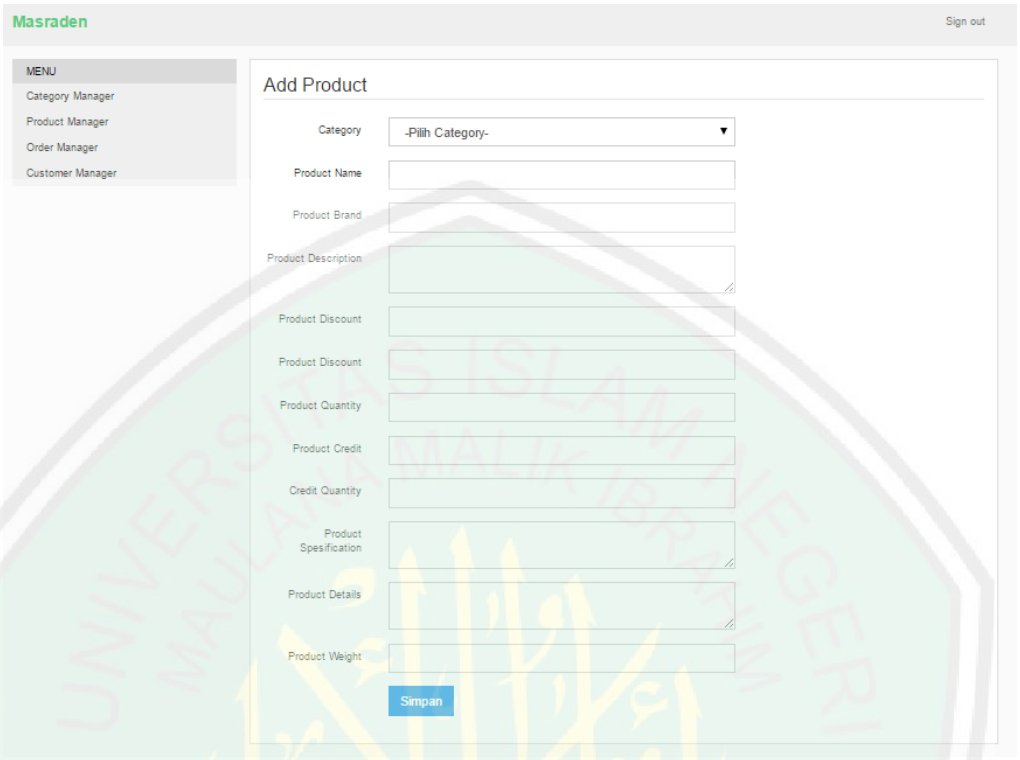
```
//===========Start Product Manager=====================
@RequestMapping(value = "/product", method = RequestMethod.GET)
public String product(Model model) {
    model.addAttribute("products", productService.getAllProducts());
    return "ManagerProduct";
}

@RequestMapping(value = "/product/add", method = RequestMethod.GET)
public ModelAndView productAdd(Model model) {
    ModelAndView mav = new ModelAndView("ManagerProductAdd");
    mav.addObject("action", "add");
    mav.addObject("product", new Product());
    mav.addObject("categorys", productService.getAllCategory());
    return mav;
}

@RequestMapping(value = "/product/add", method = RequestMethod.POST)
public String productAddPost(Model model, @ModelAttribute("product") @Valid Product product,
        BindingResult result, RedirectAttributes attributes) {
    if (result.hasErrors()) {
        model.addAttribute("action", "add");
        model.addAttribute("categorys", productService.getAllCategory());
        return "ManagerProductAdd";
    }
    productService.addProduct(product);
    String message = "Hi, " + product.getProductName() + ". You are successfully added!";
    attributes.addFlashAttribute("message", message);
    return "redirect:/manager/product";
}
```

Gambar 4.28. *listing code* manajemen barang pada *ManagerController*

Adapun tampilan halaman manajer produk sebagaimana dapat dilihat pada gambar berikut ini.



Gambar 4.29. Tampilan halaman manajemen barang



Gambar 4.30. Tampilan *form* halaman manajemen katalog

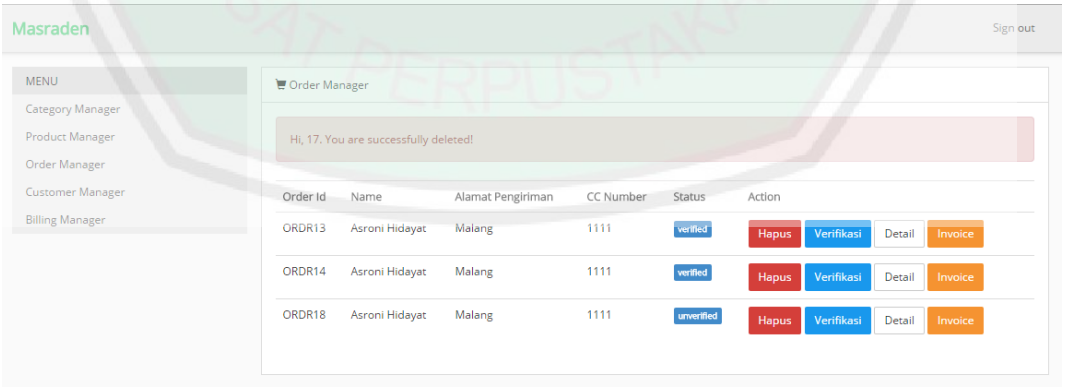
4.3.3.10. Implementasi Manajemen Pemesanan (*order*)

Proses untuk menampilkan daftar *order* adalah pada menu manajemen *order*. *Administrator* mengakses melalui *path url /manager/order* maka akan mengakses *ManagerController* dengan fungsi *order()*. *Administrator* memiliki akses untuk menghapus, memverifikasi order, melihat detail *order* dan *invoice*. Dalam proses verifikasi sistem akan memvalidasi dan melakukan penagihan *(billing)* dengan kartu kredit yang digunakan. Berikut implementasi *listing code* manajemen *order* pada *ManagerController* sebagaimana dapat dilihat pada gambar 4.30.

```
//---------Start Order Manager----------------//
@RequestMapping(value = "/order", method = RequestMethod.GET)
public String order(Model model) {
    model.addAttribute("orders", orderService.getAllOrder());
    return "ManagerOrder";
}

@RequestMapping(value = "/order/delete/{id}", method = RequestMethod.GET)
public String orderDelete(@PathVariable(value = "id") String id,
        RedirectAttributes attributes) {
    orderService.deleteOrder(id);
    String message = "Hi, " + id + ". You are successfully deleted!";
    attributes.addFlashAttribute("message", message);
    return "redirect:/manager/order";
}

@RequestMapping(value = "/order/detail/{id}", method = RequestMethod.GET)
public String orderDetail(Model model, @PathVariable(value = "id") String id) {
    model.addAttribute("orders", detailOrderService.detailOrder(id));
    return "ManagerOrderDetail";
}

@RequestMapping(value = "/order/invoice/{id}", method = RequestMethod.GET)
public String orderInvoice(Model model, @PathVariable(value = "id") String id) {
    model.addAttribute("orders", detailOrderService.detailOrder(id));
    return "ManagerOrderInvoice";
}
//---------End Order Manager----------------//
```

Gambar 4.31. *listing code* manajemen pemesanan pada *ManagerController*

Adapun tampilan halaman manajemen *order* sebagaimana dapat dilihat pada gambar 4.31.



Gambar 4.32. Tampilan halaman manajemen pemesanan

4.3.3.11. Implementasi Manajemen Penagihan (*billing*)

Setelah admin login, *administrator* dapat mengakses manajemen penagihan yang disana terdapat fitur *invoice* dari setiap transaksi yang ada. Fitur tersebut terdapat pada method *billingInvoice()* yang terletak pada class *ManagerControlle*r. Berikut implementasi *listing code* manajemen penagihan pada *ManagerControlle*r sebagaimana dapat dilihat pada gambar 4.32.

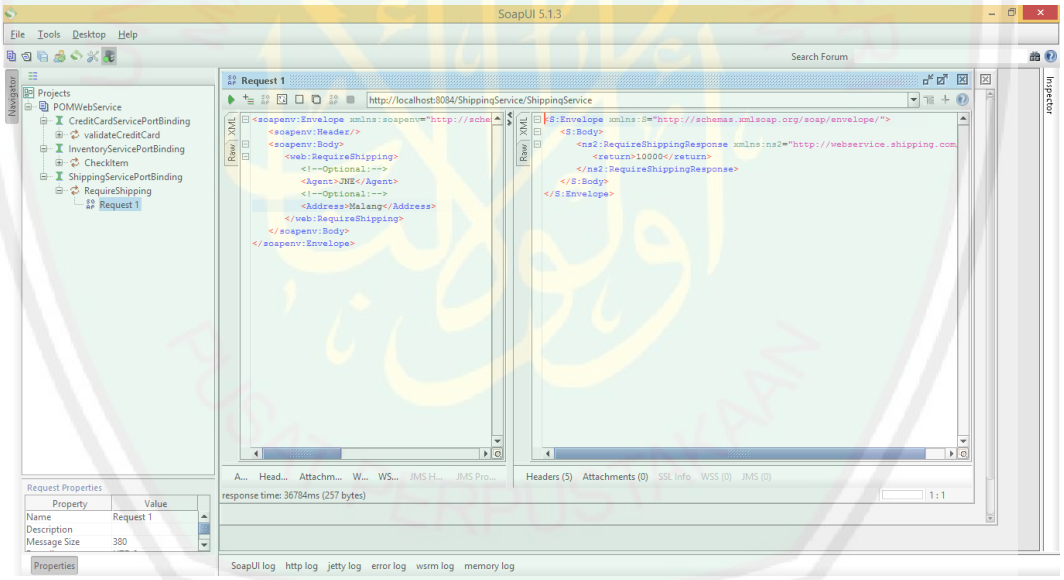
```
@RequestMapping(value = "/billing/invoice/{id}", method = RequestMethod.GET)
public String billingInvoice(Model model, @PathVariable(value = "id") Integer id) {
    model.addAttribute("shipping", productOrderService.getShippingAddress(id));
    model.addAttribute("payment", productOrderService.getPayment(id));
    // model.addAttribute("customer", productOrderService.getCustomer(id));
    model.addAttribute("order", productOrderService.getOrder(id));
    model.addAttribute("items", detailOrderService.detailOrder(String.valueOf(id)));
    return "ManagerOrderInvoice";
}
```

Gambar 4.33. *listing code* manajemen penagihan.

Gambar 4.34. Tampilan manajemen penagihan.

4.4. Pengujian Sistem

Tahapan yang terakhir dalam pengembangan sistem adalah proses pengujian sistem secara menyeluruh. Pengujian pertama yaitu adalah menguji fungsi-fungsi yang ada dalam *web service*. Apakah sudah berjalan dengan baik atau tidak, pada pengujian ini menggunakan *software testing* untuk *web service* yaitu soapUI 5.1.3. Pengujian ini dilakukan untuk mengecek kesesuain parameter, fungsi serta operasi telah berjalan dengan baik dan sesuai dengan yang diharapkan. Berikut tampilan soapUI dalam proses *testing web service* pada gambar 4.34.



Gambar 4.35. Tampilan *software* soapUI untuk *testing web service*

Hasil pengujian *web service* POM sistem *e-commerce* dengan menggunakan *software* soapUI dijelaskan secara detail pada tabel 4.1.

Tabel 4.2. Hasil pengujian *web service* POM

No	Nama Operasi	Parameter Input	Output	Status
1	validateCreditCard()	5100908903982009, Suryono, 12, 15, 590, 10000000	*valid*	sesuai
2	setCreditAmount()	5100908903982009, 2000000	*success*	sesuai
3	CheckShipping()	TIKI, Malang	17000	sesuai
4	CheckItem()	2101,2	*valid*	sesuai
5	SetQtyItem()	2101, 2	*success*	sesuai
6	calculatePayment()	1500000,2,10000	3010000	sesuai

Pengujian yang kedua pada penelitian ini yaitu pengujian pada sistem dengan menggunakan metode *blackbox*. *Black box* merupakan metode pengujian yang memfokuskan pada fungsionalitas dan hasil keluaran dari proses sebuah sistem. Skenario pengujian mengacu pada *use case* dalam sistem. Berikut tabel pengujian fungsionalitas sistem *e-commerce* dapat dilihat pada tabel 4.3:

Tabel 4.3. Tabel Pengujian Sistem

No	Nama *use case*	Kriteria Evaluasi Hasil
1	Melihat Barang di katalog	Sistem menampilkan halaman katalog, menampilkan barang berdasarkan kategori dan menampilkan detail barang
2	Registrasi	Sistem menampilkan form registrasi, validasi data registrasi dan menyimpan data registrasi.
3	Login	Sistem memeriksa apakah username, password sudah benar
4	Menambah ke keranjang	Sistem menampilkan halaman kelola keranjang belanja
5	Checkout	Sistem menampilkan form shipping dan payment serta memvalidasi proses checkout
6	Melihat daftar pemesanan	Sistem menampilkan riwayat pemesanan beserta detailnya
7	Merubah data akun	Sistem menampilkan form merubah data aku, memvalidasi data dan update data
8	Manajemen pelanggan	Sistem menampilkan *list* pelanggan. Admin dapat melakukan CRUD data pelanggan.
9	Manajemen pemesanan	Sistem dapat menampilkan *list* pemesanan. Admin dapat melakukan CRUD data pemesanan.
10	Manajemen barang	Sistem menampilkan *list* barang. Admin dapat melakukan CRUD data barang.
11	Manajemen penagihan	Sistem menampilkan *list* tagihan dan invoice.

Skenario ini dilakukan oleh pengguna dengan mengakses sistem melaui *web server local* (*localhost*). Setelah itu pengguna mengisi kuesioner yang telah diberikan. Pengujian terdiri dari pengujian fungsionalitas sistem dan antarmuka. Pengujian ini melibatkan 10 responden dari berbagai kalangan. Berikut hasil pengujian fungsionalitas sistem dan antarmuka sebagaimana terlihat pada tabel 4.4 dan tabel 4.5

Tabel 4.4. Tabel pengujian fungsionalitas sistem

No	Nama *use case*	SS	S	TS	STS
1	Sistem menampilkan halaman katalog, menampilkan barang berdasarkan kategori dan menampilkan detail barang	7	3	-	-
2	Sistem menampilkan form registrasi, validasi data registrasi dan menyimpan data registrasi.	2	8	-	-
3	Sistem memeriksa apakah username, password sudah benar	5	5	-	-
4	Sistem menampilkan halaman kelola keranjang belanja	6	4	-	-
5	Sistem menampilkan form shipping dan payment serta memvalidasi proses checkout	2	8	-	-
6	Sistem menampilkan riwayat pemesanan beserta detailnya	4	6	-	-
7	Sistem menampilkan form merubah data akun, memvalidasi data dan update data	2	8	-	-
8	Sistem menampilkan *list* pelanggan. Admin dapat melakukan CRUD data pelanggan.	4	6	-	-
9	Sistem dapat menampilkan *list* pemesanan. Admin dapat melakukan CRUD data pemesanan.	5	5	-	-
10	Sistem menampilkan *list* barang. Admin dapat melakukan CRUD data barang.	6	4	-	-
11	Sistem menampilkan *list* tagihan dan invoice.	6	4	-	-
Total		49	61		

Tabel 4.5. Tabel pengujian antarmuka sistem

No	Pernyataan	SS	S	TS	STS
1	Layout antarmuka *user friendly* sehingga mempermudah user dalam menggunakan sistem	3	6	1	-
2	Pesan validasi form sangat mempermudah dalam proses menginput data	1	9		-
3	Menu navigasi mudah dipahami	4	5	1	-
4	Menu navigasi berfungsi dengan baik	3	5	2	-
	Total	11	25	4	-

Dari hasil pengujian sistem yang diketahui melalui tabel 4.4, diperoleh hasil yang menyatakan bahwa pengguna setuju terhadap fungsionalitas sistem yang dibuat. Dari data pada tabel 4.4 didapatkan hasil pengujian fungsional sistem yang memperlihatkan bahwa 44.5% pengguna menyatakan sangat setuju bahwa sistem telah berfungsi dengan baik; sedangkan 55.5% pengguna menyatakan setuju. Selanjutnya, dalam pengujian antarmuka dan pengaksesan sistem berdasarkan hasil pada tabel 4.5 menunjukkan bahwa 27.5% pengguna sangat setuju bahwa antarmuka dan proses pengaksesan sistem mudah dan berjalan dengan baik; sedangkan 62.5% menyatakan setuju; 10% menyatakan tidak setuju.

Berdasarkan kesesuaian hasil pengujian *web service* pada tabel 4.3 dapat diambil kesimpulan bahwa sistem *e-commerce* dengan SOA telah bisa mengatasi permasalahan *loosely coupled* dan *interoperability* sistem. Selanjutnya, berdasarkan hasil pengujian sistem pada tabel 4.4 dan 4.5 sistem *e-commerce* yang telah dibuat sudah layak untuk digunakan. Tetapi perlu adanya pengembangan yang

lebih serius terkait dengan desain tampilan, level integrasi dan keamanan sistem untuk memperoleh hasil yang lebih baik dan lebih maksimal.

4.5. E-Commerce dalam perspektif keislaman

Islam selalu menyentuh semua aspek dalam kehidupan ini. Tak terkecuali dengan muamalah atau berbisnis. Islam telah memberikan batasan yang sangat jelas tentang berdagang, sebagaimana firman Allah SWT surat Al-Baqarah ayat 275 :

وَأَحَلَّ اللهُ الْبَيْعَ وَحَرَّمَ الرِّبَا

Allah telah menghalalkan jual beli dan mengharamkan riba" (Al-Baqarah:275)

Bermuamalah bukan semata-mata masalah transaksi uang dan barang, akan tetapi juga masalah etika dan akhlak, yang semuanya telah diatur dalam Islam dengan Al-Qur'an dan sunnah-Nya. Maka kuncinya, harus dipastikan semua proses dalam transaksi muamalah sesuai dengan hukum islam, tidak merugikan orang lain, jujur, transparan, tidak riba, amanah dan lain sebagainya. Berikut adalah dalil-dalil yang menjelaskan tentang *e-commere* yang peneliti kutip dari jurnal yang berjudul *"E-Commerce from an Islamic perspective"* .

1. Etika berisnis dalam Islam

Salah satu bentuk muamalah dalam dunia modern sekarang adalah e-commerce. Dalam sudut pandang islam, *e-commerce* (perdagangan elektronik) memiliki definisi yang sama dengan *conventional commerce* (perdagangan konvensional). Dalam menjalankannya haruslah sesuai dengan prinsip-prinsip keislaman. Sebagaimana firman Allah dalam surat Al-Jumu'ah ayat 10

فَإِذَا قُضِيَتِ ٱلصَّلَوٰةُ فَٱنتَشِرُوا۟ فِى ٱلْأَرْضِ وَٱبْتَغُوا۟ مِن فَضْلِ ٱللَّهِ
وَٱذْكُرُوا۟ ٱللَّهَ كَثِيرًا لَّعَلَّكُمْ تُفْلِحُونَ ﴿١٠﴾

"Apabila telah ditunaikan shalat, Maka bertebaranlah kamu di muka bumi; dan carilah karunia Allah dan ingatlah Allah banyak-banyak supaya kamu beruntung" (QS. Al-Jumu'ah: 10)

2. Legitimasi kontrak dalam *e-commerce*

Islam mendefinisikan kontrak sebagai kesepakatan dan persetujuan/perjanjian antara kedua belah pihak dalam hal jual beli. Kontrak terjadi dengan jalan ijab kabul terlebih dahulu. Setelah kontrak sah, maka penjual wajib mematuhi janji yang sudah disepati dengan pembeli, begitu pula sebaliknya. Hal ini sesuai dengan firman Allah SWT surat Al-Maidah ayat 1 :

يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوٓا۟ أَوْفُوا۟ بِٱلْعُقُودِ

"Wahai orang-orang yang beriman! Penuhilah janji-janji". (QS. Al-Maidah : 1)

3. Validitas transaksi *e-commerce*

Dalam proses transaksi haruslah bebas dari yang namanya riba. Riba didefinisikan sebagai tambahan nilai yang diisyaratkan dalam transaksi bisnis tanpa adanya iwadh (atau padanan) yang dibenarkan syariah atas penambahan tersebut. Islam melarang riba, sebagaimana dalam Al-Quran surat Ali Imran 130.

يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ لَا تَأْكُلُوا۟ ٱلرِّبَوٰٓا۟ أَضْعَٰفًا مُّضَٰعَفَةً ۖ
وَٱتَّقُوا۟ ٱللَّهَ لَعَلَّكُمْ تُفْلِحُونَ ﴿١٣٠﴾

*"Hai orang-orang yang beriman, janganlah kamu memakan **riba** dengan berlipat ganda dan bertakwalah kamu kepada Allah supaya kamu mendapat keberun-tungan. "* (QS. Ali Imran : 130)

4. Nilai-nilai kejujuran dan amanah dalam berdagang

Sistem *e-commerce* menyediakan informasi barang secara transparan. Mulai dari harga, spesifikasi dan lain-lain. Hal ini menjadikan *e-commerce* memberikan informasi yang jelas dan apa adanya. Sistem yang terintegrasi membuat proses transaksi perdagangan menjadi lebih cepat dan valid. Karena proses sudah terkomputerisasi dengan baik, maka bisa meminimalisir hal-hal negatif yang mungkin saja terjadi dalam transaksi manual. Setiap transaksi adalah sebuah amanah yang diberikan pelanggan maka sebagai penyedia layanan *e-commerce* haruslah melayani transaksi sesuai dengan yang diharapkan yakni jujur lagi amanah. Kerjujuran dan amanah dalam konsep berdagang dengan *e-commerce* ini sesuai dengan hadits berikut ini :

عَنْ عُبَادَةَ بْنِ الصَّامِتِ رض اَنَّ النَّبِيَّ ص قَالَ: اِضْمَنُوْا لِي سِتًّا مِنْ اَنْفُسِكُمْ، اَضْمَنْ لَكُمُ الْجَنَّةَ. اُصْدُقُوْا اِذَا حَدَّثْتُمْ، وَ اَوْفُوْا اِذَا وَعَدْتُمْ، وَ اَدُّوْا اِذَا ائْتُمِنْتُمْ، وَ احْفَظُوْا فُرُوْجَكُمْ، وَ غُضُّوْا اَبْصَارَكُمْ، وَ كُفُّوْا اَيْدِيَكُمْ. احمد و ابن ابى الدنيا و ابن حبان فى صحيحه و الحاكم و البيهقى

Dari Ubadah bin Shamit RA sesungguhnya Nabi SAW bersabda : "Hendaklah kalian menjamin padaku enam perkara dari dirimu, niscaya aku menjamin surga bagimu : 1. Jujurlah apabila kamu berbicara, 2. Sempurnakanlah (janjimu) apabila kamu berjanji, 3. Tunaikanlah apabila kamu diberi amanat, 4. Jagalah kemaluanmu, 5. Tundukkanlah pandanganmu (dari ma'shiyat) dan 6. Tahanlah tanganmu (dari hal yang tidak baik)". [HR. Ahmad, Ibnu Abid-Dunya, Ibnu Hibban di dalam shahihnya, Hakim dan Baihaqi]

Dari dalil diatas bisa diambil kesimpulan, menjadi muslim yang jujur adalah sebuah keharusan, karena sikap jujur mencerminkan integritas keislaman. Terutama dalam proses perdagangan dalam dunia modern seperti *e-commerce*. Semua proses haruslah sesuai dengan nilai-nilai kejujuran dan amanah yang telah Islam ajarkan.

BAB V

PENUTUP

5.1. Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi dan pengujian dapat diambil kesimpulan bahwa :

1. Penelitian ini berhasil mengimplementasikan sistem *e-commerce* berbasis *Service Oriented Architecture* dengan teknologi *web services* (JAX-WS Metro) dan Java EE (*Java Enterprise Edition*).
2. Penelitian ini berhasil mengembangkan sistem *e-commerce* terintegrasi pada proses *Purchase Order Management* (POM) dengan menggunakan *Service Oriented Architecture* dengan pendekatan berorientasi objek.
3. Penelitian ini berhasil mengembangkan sistem *e-commerce* terintegrasi menggunakan *Service Oriented Architecture* dalam *Purchase Order Management* (POM). Dalam proses POM, sistem berinteraksi dengan sistem yang berbeda-beda melalui *web services*. Diantaranya, sistem *Billing* (penagihan), *Shipping* (pengiriman), *Inventory* (gudang) dan *Payment* (pembayaran).

5.2. Saran

Sistem *e-commerce* yang telah dibangun ini tentunya tidak lepas dari kekurangan. Terutama dalam metode integrasi sistem, dari sebab itu peneliti menyarankan beberapa hal, diantaranya :

1. Sistem bisa dikembangkan lagi kedalam model EAI (*Enterprise Application Integration*) dalam hal mengatur pola integrasi dengan banyak *inventory* (gudang).
2. Sistem sebaiknya dilengkapi fitur *realtime* notifikasi SMS, dengan SMS *Gateway*.
3. Sistem perlu dikembangkan lagi dalam hal fitur pencarian (*searcing*) barang dengan model *elastic search* dan fitur *filter products* dengan *multiple parameter* atau *matrix variable*.
4. Selain itu adalah dalam proses pembayaran, sistem membatasi hanya dalam penggunaan kartu kredit. Mekanisme pembayaran lain seperti *paypal*, *bank transfer* dan cicilan belum dimodelkan. Penetili berharap ada yang meneruskan pengembangan ini ke level yang lebih kompleks.
5. Sistem *e-commerce* menggunakan SOA ini bisa dijadikan acuan, bukan hanya dikembangkan pada sistem *e-commerce* B2C (*Business-to-Consumer*) tapi juga B2B (*Business-to-Business*).

DAFTAR PUSTAKA

Arba, R., "*Using SOA for E-Commerce Systems and Development"* in Proceedings of The International Conference on Knowledge Engineering KEPT 2009 Cluj-Napoca, (Romania), 2009, pp. 3-6.

Rebhi S. Baraka, Yousef M. Al-Ashqar. 2013. *Building a SOA-Based Model for Purchase Order Management in E-Commerce Systems*. Palestinian International Conference on Information and Communication Technology.

Khoirul Haq, Azizi. 2012. *Rancang Bangun Sistem Informasi Apotik Terintegrasi Menggunakan Service Oriented Architecture*. Yogyakarta: UIN Sunan Kalijaga

Frisandi, Fachry. Supangkat, H.S. 2011. *Implementasi Service-Oriented Architecture Pada Pengembangan Sistem Pembelajaran Mobile.* Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung

Rifai, Ahmad. 2011. *Rancang Bangun Sistem Persediaan (Inventory) Dengan Model Software As A Service Menggunakan Service Oriented Architecture.* Teknik Infromatika, Institut Teknik Sepuluh Nopember.

Sarwosri, Farah Naja. 2011. *Rancang Bangun Perangkat Lunak Aplikasi Pelayanan Kesehatan Berbasis Service-Oriented Architecture*. Seminar Nasional Teknologi Informasi & Komunikasi Terapan 2011 (Semantik 2011)

Safuwan, Riyanarto S, Rizky J.A. 2010. *Integrasi Perangkat Lunak Enterprise Resource Planning (ERP) Dengan Menggunakan Metode Service Oriented Architecture (SOA).* Surabaya: Institut Teknologi Sepuluh Nopember

Erl, Thomas. 2005. *Service-Oriented Architecture: Concepts, Technology, and Design*. USA : Prentice Hall

Ezenwoke A., Misra S., Adigun M. *An Approach for e-Commerce On-Demand Service-oriented Product line Development*. *Acta Polytechnica Hungarica.* Vol. 10, No. 2, 2013

Juric, M.B., Loganathan, R., Sarang, P., dan Jennings, F., 2007, *SOA Approach to Integration*, Packt Publishing, Birmingham, B27 6PA, UK.

Heffelfinger, Dafid R. 2014. *Java EE 7 with GlassFish 4 Application Server*. Birmingham, UK : Packt Publishing.

Chappell, david., Jewel, Tyler. 2002. *Java Web Services.* Boston : O'Reilly

G, Amuthan. 2014. *Spring MVC Beginner's Guide.* Birmingham, UK : Packt Publishing

Schmutz, G., Welkenbach, P., Liebhart, D.. *Service Oriented Architecture: An Integration Blueprint*. Birmingham, UK : Packt Publishing

Utomo, W Herry. 2012. *Pemrograman Aplikasi SOA.* Salatiga: UKSW Repository

www.codeproject.com/Articles/674959/MVC-Patterns-Active-and-Passive-Model-and-its diakses tanggal 17 juni 2015

Zainul, N., dkk. 2004. *E-Commerce from an Islamic prespective.* Kuala Lumpur : International Islamic University Malaysia.

Wahono, R S. 2014. *System Analysis and Design.* romisatriawahono.net/sad

G Loudon, Kenneth., P Loudon, Jane. 2012. *Management Information System :Twelfth Edition.* USA : Prentice Hall

Winch, R., Mularien, P. 2012. *Spring Security 3.1.* UK : Packt Publishing

Seddighi, RA. 2009. *Spring Presistence with Hibernate.* UK : Packt Publishing

Alan Dennis dkk.2010.*Systems Analysis and Design with UML 4th Edition.* USA: John Wiley and Sons.

Toha, Mudzakkir. 2010. *Implementasi Framework Spring MVC Untuk Pembuatan Sistem Informasi Manajemen E Commerce.* Tugas Akhir, Fakultas Matematika Dan Ilmu Pengetahuan Alam Universitas Sebelas Maret, Surakarta

http://stackoverflow.com/questions/14872020/difference-between-soa-and-esb diakses tanggal 17 juni 2015

Tarmizi, Erwandi. 2012. *Fiqh Perbankan Syariah: Pengantar fiqh muamalat dan aplikasinya dalam ekonomi modern.* Fakultas Syariah Universitas Islam Imam Muhammad Saud

LAMPIRAN

Form* Pengujian Sistem *E-Commerce

Menggunakan SOA(*Service Oriented Architecture*)

Nama : .....

Instansi : .....

Tanggal : .....

Keterangan : berilah tanda √ pada salah satu kolom setiap pertanyaan dibawah ini.

No	Nama *use case*	SS	S	TS	STS
1	Sistem menampilkan halaman katalog, menampilkan barang berdasarkan kategori dan menampilkan detail barang				
2	Sistem menampilkan form registrasi, validasi data registrasi dan menyimpan data registrasi.				
3	Sistem memeriksa apakah username, password sudah benar				
4	Sistem menampilkan halaman kelola keranjang belanja				
5	Sistem menampilkan form shipping dan payment serta memvalidasi proses checkout				
6	Sistem menampilkan riwayat pemesananan beserta detailnya				
7	Sistem menampilkan form merubah data aku, memvalidasi data dan update data				
8	Sistem menampilkan *list* pelanggan. Admin dapat melakukan CRUD data pelanggan.				
9	Sistem dapat menampilkan *list* pemesananan. Admin dapat melakukan CRUD data pemesananan.				
10	Sistem menampilkan *list* barang. Admin dapat melakukan CRUD data barang.				
11	Sistem menampilkan *list* tagihan dan invoice.				

No	Pernyataan	SS	S	TS	STS
1	Layout antarmuka *user friendly* sehingga mempermudah user dalam menggunakan sistem				
2	Pesan validasi form sangat mempermudah dalam proses menginput data				
3	Menu navigasi mudah dipahami				
4	Menu navigasi berfungsi dengan baik				