UNIVERSITAS INDONESIA

INTEGRASI ONTOLOGI DAN *WEB SERVICES* PADA ZOTONIC

SKRIPSI

ANDRI KURNIAWAN
1306382064

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNI 2017

UNIVERSITAS INDONESIA

INTEGRASI ONTOLOGI DAN *WEB SERVICES* PADA ZOTONIC

SKRIPSI
Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Ilmu Komputer

ANDRI KURNIAWAN
1306382064

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNI 2017

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Andri Kurniawan**
NPM	:	**1306382064**
Tanda Tangan	:	
Tanggal	:	**5 Juni 2017**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Andri Kurniawan
NPM	:	1306382064
Program Studi	:	Ilmu Komputer
Judul Skripsi	:	Integrasi Ontologi dan *Web Services* pada Zotonic

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing	:	Dr. Ade Azurat	(	)
Penguji	:	Penguji 1	(	)
Penguji	:	Penguji 2	(	)

Ditetapkan di	:	Depok
Tanggal	:	5 Juli 2017

KATA PENGANTAR

Segala puji dan syukur penulis ucapkan atas kehadirat Allah SWT, Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya penulis dapat menyelesaikan skripsi yang berjudul "Integrasi Ontologi dan *Web Services* pada Zotonic". Pada kesempatan ini penulis ingin mengucapkan terima kasih kepada seluruh pihak yang telah membantu dan mendukung penulis selama proses pengerjaan skripsi ini, dimana berkat dukungan dan doa mereka skripsi ini dapat diselesaikan.

Penulisan skripsi ini ditujukan untuk memenuhi salah satu syarat untuk menyelesaikan pendidikan pada Program Sarjana Ilmu Komputer, Universitas Indonesia. Penulis sadar bahwa dalam perjalanan perkuliahan hingga penulisan skripsi ini, penulis tidak sendirian. Penulis ingin berterima kasih kepada pihak-pihak berikut :

1. Bapak Ade Azurat selaku dosen pembimbing tugas akhir yang telah meluangkan waktunya sepanjang semester ini untuk dapat memberikan arahan, kritik dan saran kepada penulis agar dapat menyelesaikan proses pengerjaan skripsi ini.

2. Bapak Drs. Lim Yohanes Stefanus M.Math., Ph.D selaku dosen pembimbing akademis penulis, yang selalu membantu penulis selama masa perkuliahan.

3. Drs. H. Zulhaspan, MM dan Hj. Masreni Nasution selaku orangtua dari penulis serta Anita Putri dan Akbar Syarif selaku saudara dari penulis yang selalu mendoakan, mendukung serta menjadi motivasi penulis dalam mengerjakan skripsi ini.

4. Kak Afifun yang telah memberikan banyak masukan terkait hal teknis kepada penulis.

5. Lab RSE

6. Nabila Akiti Hara selaku pacar dari penulis yang selalu memberikan motivasi dan mendukung penulis selama pengerjaan skripsi serta menemani penulis melalui *facetime*.

7. Teman-teman PI BPH IMMM UI (Fadhil, Titto, Fandika, Mawan, Dodo, Okky, Devi, Rara, Ami, Rizky, Ime, Popo, Ilham) yang selalu menghibur

penulis kala jenuh dalam mengerjakan skripsi dan seluruh keluarga IM-MMSU UI yang telah menjadi keluarga bagi penulis selama masa perkuliahan.

8. Arief Radityo, Arsi Alhafis, dan M. Gibran yang selalu menjadi teman untuk bermain maupun belajar bagi penulis serta membantu penulis selama perkuliahan.

9. Sahabat-sahabat PPN (Abi, Budi, Cia, Dana, Erwin, Fakhry, Irene, Fadly, Fani, Mawan, Mutia, Sufi, Ulup) yang selalu menjadi penghibur bagi penulis setiap saat.

10. Teman-teman CornedIn (Arsi, Zaki, Dimas, Ilham) yang merupakan teman-teman perjuangan untuk proyekan yang mengajarkan banyak hal terkait teknikal kepada penulis.

11. Kelompok PPL B1 (Akbar, Dimas, Emon, Fajrin, Fathin), Kelompok PPL B2 (Gilang, Falah, Fatah, Nanda, Hamdan) dan Kelompok PPL B3 (Brigita, Gentur, Kowan, Riscel, Muthy) serta Kak Naya yang telah menemani penulis selama satu semester khususnya hari Rabu dan memberikan penulis pandangan baru mengenai *scrum master*.

Akhir kata, penulis berharap semoga Allah SWT dapat membalas kebaikan yang diberikan oleh orang-orang terdekat penulis dan penulis berharap karya yang penulis buat dapat membantu dan bermanfaat bagi pengembangan ilmu pengetahuan selanjutnya.

Depok, 5 Juni 2017

Andri Kurniawan

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Andri Kurniawan
NPM	:	1306382064
Program Studi	:	Ilmu Komputer
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

Integrasi Ontologi dan *Web Services* pada Zotonic

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 5 Juni 2017

Yang menyatakan

(Andri Kurniawan)

ABSTRAK

Nama	:	Andri Kurniawan
Program Studi	:	Ilmu Komputer
Judul	:	Integrasi Ontologi dan *Web Services* pada Zotonic

Abstrak INA

Kata Kunci:
ABS, Adaptor, SPL, *Web Service*, Zotonic

ABSTRACT

Name	:	Andri Kurniawan
Program	:	Computer Science
Title	:	Ontology and Web Services Integration on Zotonic

Abstract in Eng

Keywords:
one,two,three

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB 1
PENDAHULUAN

@todo
Buat penjelasan singkat di Bab 1 ada apa aja

1.1 Latar Belakang

Pada zaman serba teknologi saat ini, kebutuhan akan penggunaan web sebagai sarana berbagi informasi sangat tinggi. Namun, tidak ada sumber daya manusia yang memiliki latar belakang teknis dan merasa sulit untuk membuat sebuah web menjadi sebuah permasalahan yang dihadapi oleh beberapa organisasi di Indonesia terutama organisasi yang non-profit. Selain itu, banyaknya informasi yang terdapat pada web saat ini tidak memiliki hubungan informasi yang terstruktur dan hanya didesain untuk manusia saja sehingga program komputer tidak dapat mengolah informasi tersebut (Berners-Lee, Hendler, dan Lassila, 2001, p. 1). Sehingga pada tahun 2001 dicetuskan ide untuk membuat web semantik oleh Berners-Lee dkk, dimana web semantik akan membuat sebuah struktur untuk konten web sehingga informasi yang terdapat pada web lebih berarti karena dapat diolah oleh program komputer (Berners-Lee, Hendler, dan Lassila, 2001, p. 1). Namun, perkembangan web semantik sendiri tidak seperti yang diharapkan karena mengalami permasalahan. Menurut Rob McCool, Format yang kompleks dan pengguna harus mengorbankan kemudahan ekspresivitas dan membayar biaya yang besar untuk translasi dan perawatan, web semantik tidak akan pernah diadopsi public secara luas (Schoop, de Moor, dan Dietz, 2006).

Untuk mengatasi permasalahan tersebut yang dihadapi oleh web semantik, diciptakan web pragmatis. Tujuan utama dari web pragmatis adalah untuk meningkatkan kolaborasi manusia lebih efektif dengan teknologi yang tepat, seperti sistem untuk negosiasi ontologi, untuk interaksi bisnis berbasis ontologi, dan untuk membangun ontologi pragmatis pada praktik masyarakat (Schoop, de Moor, dan Dietz, 2006). Sehingga web pragmatis dapat melengkapi web semantik untuk berkolaborasi dan meningkatkan kualitas pada level masyarakat. Salah satu perkembangan web semantik pragmatis adalah Zotonic, yaitu sebuah framework sekaligus Content Management System (CMS) yang dibangun di atas bahasa pemrogra-

man erlang dimana zotonic telah mengadopsi konsep web semantik. Kehadiran Zotonic sendiri diharapkan dapat meningkatkan pemanfaatan web semantik dalam proses pembuatan web sehingga infomasi yang berada pada web yang dibuat dapat langsung diolah oleh komputer. Tetapi, pengembang perlu mendefinisikan semantik yang akan mereka buat terlebih dahulu sebelum mereka mengembangkannya dalam Zotonic untuk menciptakan sebuah konsistensi. Namun hal ini tentu saja menghambat proses pengembangan karena pengembang membutuhkan waktu yang lebih lama untuk proses translasi dari semantik ke Zotonic.

Untuk membantu para pengembang dalam mentranslasikan semantik ke dalam Zotonic, pada tahun 2016 terdapat sebuah penelitian terkait pembentukan otomatis aplikasi web dengan masukan berupa ontologi, yaitu penelitian terkait pemetaan ontologi yang dihasilkan semantik ke dalam struktur Zotonic (Pangukir el al., 2016). Namun, pada penelitian tersebut pembentukan bussiness logic dari web yang dibentuk masih secara manual. Hal ini karena pada ontologi tidak terdapat bussiness logic. Bussiness logic tersebut dapat diperoleh melalui translasi feature model dari rancangan web tersebut.

Oleh karena itu, untuk menerapkan paradigma Software Product Line (SPL) dimana variasi dan kesamaan akan disusun secara modular dan reusability, diperlukan sebuah program yang akan melakukan pemetaan secara otomatis terhadap bussiness logic dari feature model yang telah dirancang ke dalam Zotonic. Sehingga diharapkan kedepannya, proses pembentukan web berbasis semantik dapat lebih mudah dan cepat agar meningkatkan jumlah web yang berbasis semantik.

@todo
Perdalam lagi latar belakang.

1.2 Perumusan Masalah

Berdasarkan latar belakang tersebut, penelitian ini akan mencoba untuk menjawab beberapa pertanyaan penelitian, yaitu

1. Bagaimana proses pemetaan dari ontologi kepada *web service*?
2. Apakah dapat dikembangkan sebuah program *adaptor* yang akan melakukan pemetaan dari zotonic kepada *web service* secara otomatis?

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk mengembangkan sebuah program yang dapat menghubungkan antara *web services* dan Zotonic sehingga proses pembuatan *site* pada Zotonic dapat memiliki tingkat adaptasi yang baik terhadap perubahan yang terjadi. Harapannya melalui program yang dibuat, pengembang dapat tetap fokus untuk *maintain* data dan mengembangkan aplikasi, karena data sudah terjadi melalui ontologi.

1.4 Ruang Lingkup Penelitian

Ruang lingkup penilitian ini antara lain:

1. Analisis pemetaan adaptor untuk menghubungkan antara ontologi dan *web service*
2. Perancangan sistem yang dapat menghubungkan antara ontologi dan *web service* melalui zotonic.
3. *Refactoring* pembuatan *business logic* yang sudah ada dengan memanfaatkan sistem yang dibuat

1.5 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN

  Bab 1 berisi tentang informasi terkait penelitian yang dilakukan oleh penulis, dimana bab ini terdiri atas 5 subbab, yaitu latar belakang yang akan membahas kenapa dilakukan penelitian ini, perumusan masalah yang akan membahas masalah yang akan diteliti oleh penulis, tujuan penelitian yang akan menjelaskan kegunaan dari penelitian ini, ruang lingkup penelitian yang akan membahas mengenai batasan dari peneletian yang dilakukan serta sistematika penulisan.

- Bab 2 TINJAUAN PUSTAKA

  Bab 2 berisi mengenai teori-teori yang digunakan

- Bab 3 RANCANGAN

  Bab 3 berisi penjelasan mengenai rancangan dari sistem yang akan diimplementasikan, dimana bab ini terdiri atas 2 subbab, yaitu rancangan integrasi ontologi dan web service yang akan membahas secara garis besar bagaimana program ini akan berjalan secara keseluruhan serta rancangan *adaptor* yang akan membahas bagaimana *adaptor* yang dibuat nantinya akan bekerja.

- Bab 4 IMPLEMENTASI

  Bab 4 berisi penjelasan mengenai impelementasi yang dilakukan oleh penulis untuk membuat *adaptor*.

- Bab 5 HASIL

@todo
Jelasin bab 5 ngapain.

  Bab 5 berisi hasil eksperimen yang telah dilakukan oleh penulis.

- Bab 6 PENUTUP

  Bab 6 berisi kesimpulan yang didapat dari penelitian serta saran yang diajukan untuk penelitian berikutnya.

BAB 2
TINJAUAN PUSTAKA

Bab ini berisi tentang tinjauan pustaka yang terkait dengan penelitian. Bab ini akan menjelaskan mengenai ontologi, zotonic, dan *software product line*.

2.1 Ontologi

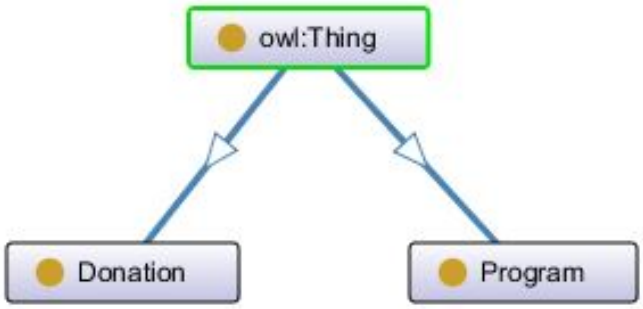
Ontologi merupakan bagian yang paling mendasar dari web semantik. Ontologi adalah sebuah deskripsi formal yang eksplisit tentang kelas (domain atau disebut juga sebagai *concepts*), relasi, dan *property* dari setiap domain, yang biasanya telah terdefinis secara baik (Hopkins dan Powell, 2015). Menurut Noy dan McGuinness (2001), pengembangan ontologi perlu dilakukan karena beberapa alasan berikut ini.

1. Berbagi pemahaman umum terkait struktur dari informasi diantara manusia ataupun *software agents*.
2. Menggunakan kembali pengetahuan dari kelas.
3. Membuat asumsi terkait kelas tersampaikan secara eksplisit.
4. Memisahkan pengetahuan tentang kelas dari pengetahuan secara operasional.
5. Menganalisa pengetahuan tentang kelas.

Berbagi pemahaman umum terkait struktur dari informasi merupakan satu dari beberapa tujuan utama pada pengembangan ontologi (Gruber, 1993). Dengan adanya pemahaman terkait struktur dari informasi yang diperoleh, sebuah *software agents* dapat mengekstrak dan mengumpulkan informasi yang diperoleh dari berbagai layanan yang berbeda namun menggunakan dasar ontologi yang sama. Selain dapat memperoleh informasi, dengan menggunakan struktur ontologi yang cenderung sama maka sebuah ontologi dapat digunakan kembali untuk kasus yang cenderung sama atau mirip sehingga dapat melakukan penghematan dana maupun waktu.

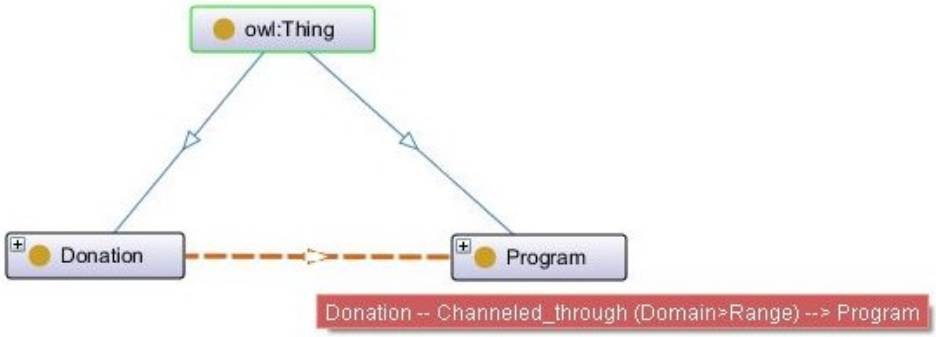
Ontologi terdiri dari beberapa komponen yaitu kelas, atribut, relasi. Ketika membuat ontologi, kelas merupakan komponen yang harus dibuat pertama kali karena kelas merupakan komponen utama dari ontologi. Kelas atau disebut juga domain, merupakan kumpulan data pada ontologi yang memiliki sebuah nama yang

deskriptif untuk menggambarkan data tersebut. Salah satu contoh dari kelas adalah sebagai berikut



Gambar 2.1: Contoh Kelas pada Ontologi

Pada Gambar 2.1, Program dan *Donation* merupakan salah satu contoh dari kelas (disebut sebagai *thing* pada *software* protege). Program merupakan kumpulan data mengenai kegiatan yang dilakukan sedangkan *Donation* merupakan kumpulan data mengenai donasi terhadap suatu program. Kelas Program dan *Donation* membutuhkan atribut yang akan menjelaskan tentang kelas tersebut sehingga perlu didefinisikan apa yang menjadi atribut dari kelas tersebut. Pada penelitian ini, penulis menggunakan ontologi yang sudah ada yaitu ontologi *charity organization* dimana kelas Program memiliki atribut name dan total sedangkan kelas *Donation* memiliki atribut *amount*. Pada gambar Gambar 2.1, belum terdapat keterhubungan diantara kelas Program dan *Donation* sehingga perlu membuat komponen terakhir yaitu relasi. Setiap relasi harus dapat menggambarkan keterhubungan dari kelas yang ada. Pada ontologi *charity organization* yang digunakan, kelas Program dan *Donation* dihubungkan menggunakan relasi *Channeled through* seperti gambar berikut

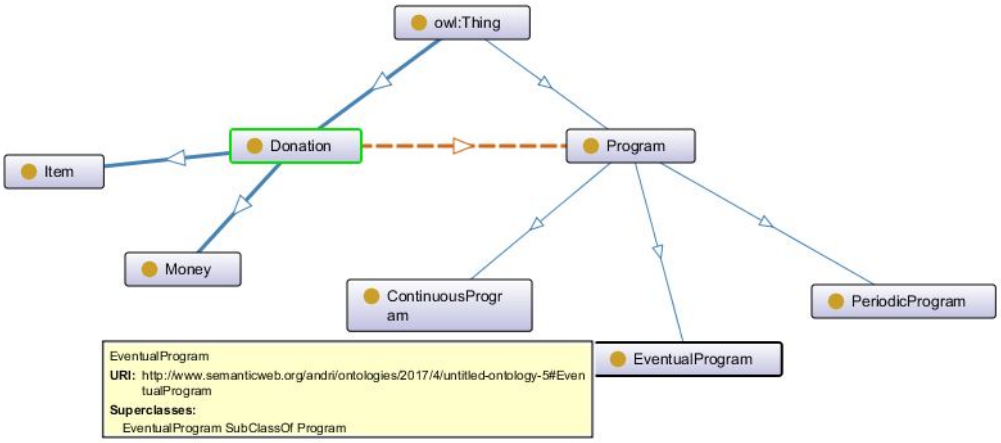


Gambar 2.2: Contoh Relasi pada Ontologi

Pada gambar Gambar 2.2, dapat dilihat bahwa kelas *Donation* dan kelas Program dihubungkan oleh relasi *Channeled through* yang berarti suatu donasi dapat

disalurkan kepada suatu program. Relasi antar keduanya digambarkan sebagai sebuah edge pada graf ontologi.

Selain memiliki tiga komponen yang telah disebutkan diatas, pada ontologi juga terdapat struktur hierarki yang dapat dimiliki oleh suatu kelas. Struktur hierariki ini menggambarkan tingkatan dari suatu kelas. Kelas yang memiliki tingkatan lebih tinggi biasanya merupakan kelas yang bersifat lebih umum dari kelas-kelas yang memiliki tingkatan lebih rendah dari dirinya. Sebagai contoh, kelas Hewan memiliki tingkatan yang lebih daripada kelas Mamalia dan juga kelas Reptil karena mamalia dan reptil merupakan bagian dari hewan. Hal ini dapat dilihat pada Gambar 2.3 dimana kelas *EventualProgram* memiliki tingkatan yang lebih rendah dari kelas Program atau disebut dengan *subclass*. Untuk kelas yang memiliki tingkatan yang lebih rendah dari kelas diatasnya, kelas tersebut akan mewarisi semua atribut atau properti yang terdapat pada kelas yang lebih tinggi dari dirinya. Sebagai contoh, kelas Program memiliki atribut *name* dan total sehingga kelas *ContinuousProgram, EventualProgram serta PeriodicProgram* akan memiliki atribut *name* dan total juga.



Gambar 2.3: Contoh Struktur Hierarki

Agar ontologi dapat dipahami oleh komputer, ontologi tersebut harus direpresentasikan dalam bentuk yang dapat dibaca oleh komputer salah satunya adalah OWL. *Ontoloy Web Language* (OWL) merupakan versi RDFS yang memiliki kosa kata dan *rules* yang lebih ketat sehingga OWL lebih mudah dipahami oleh komputer karena dapat lebih menggambarkan ontologi yang dibuat (OWL-Working-Group, 2004). Pada penelitian ini, digunakan setidaknya tiga elemen dari OWL yaitu *class, datatype property* serta *object property*. *Class* akan menggambarkan kelas dari ontologi, *datatype property* akan menggambarkan atribut dari kelas, serta *object property* akan menggambarkan relasi antar suatu kelas dengan kelas lainnya.

2.2 Web *Framework* Berbasis Semantic: Zotonic

@todo
Jelasin dahulu apa itu web semantik

Zotonic merupakan web yang bersifat *open source*, *real-time web framework*, dan sekaligus sebagai *Content Management System* (CMS) yang dibangun menggunakan bahasa pemrograman erlang (Zotonic, nda). Selain merupakan sebuah CMS dan *framework*, Zotonic juga merupakan sebuah web server yang dapat menjalankan web langsung tanpa bantuan web server seperti Apache dan Nginx. Karena dibangun dengan bahasa pemrograman erlang, Zotonic sangat mengutamakan kecepatan.

Menurut Zotonic, kecepatan dari zotonic ini sendiri bisa mencapai 10x lebih cepat daripada CMS yang dibangun menggunakan bahasa pemrograman PHP. Hal ini berdasarkan perbandingan antara proses pembuatan halaman pada kebanyakan *framework* PHP yang membutuhkan waktu 150-600 ms sedangkan pada Zotonic biasanya hanya membutuhkan waktu 10 ms atau kurang. Selain itu, Zotonic memiliki mekanisme untuk mencegah permintaan yang berbeda untuk melakukan hal yang sama pada waktu yang bersamaan. Ketika terdapat dua atau lebih permintaan datang untuk halaman yang sama, atau bagian dari halaman yang sama, maka zotonic akan melakukan pekerjaan sekali dan mengirimkan hasilnya ke semua permintaan yang masuk. Zotonic juga menyimpan data yang sering digunakan pada memori, sehingga dapat mencegah kueri yang banyak kepada database. Pada erlang, kode akan dimuat dan akan terus dimuat sampai ada perubahaan kode berikutnya sehingga hal ini dapat membuat zotonic lebih efisien dibandingkan PHP (Zotonic, ndc).

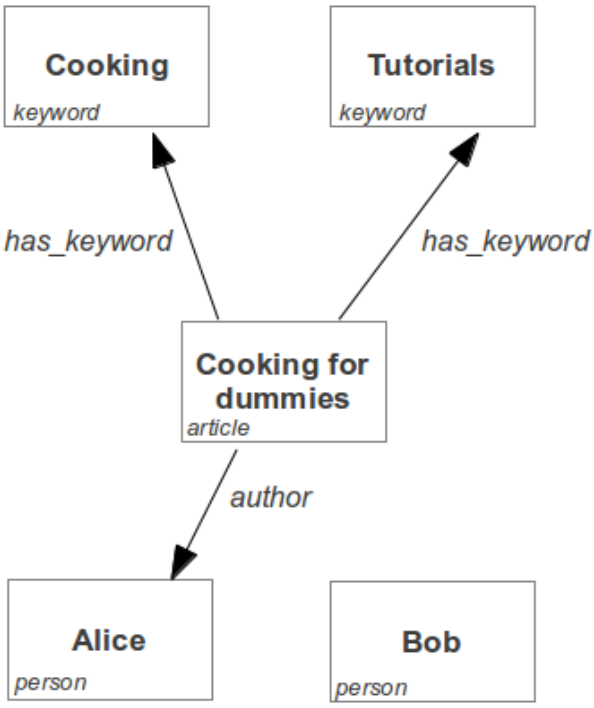
Zotonic juga menerapkan sistem modular. Zotonic dibuat dari lapisan kode yang umum dengan modul yang menyediakan fungsi utama lainnya. *Template, actions, controllers, javascript, css, dispatch rules*, semuanya berasal dari modul. Modul dapat memperbanyak, mengubah atau bekerja sama dengan modul lainnya. Modul dapat mendefinisikan ulang *template, javascript, css, actions* dan *dispatch rules* dari modul lainnya.

Zotonic juga melakukan pemisahan antara model, tampilan, dan *controller* atau biasa disebut MVC yang telah menjadi *best practice* pada proses pembuatan web dalam waktu yang lama. Hal ini bagus untuk pemisahan tanggung jawab (Zotonic, ndb). Hal ini akan memberikan kesempatan kepada *programmer* dari program dan pengembang tampilan (*front-ender*) untuk membuat HTML, CSS, dan lainnya sendiri berdasarkan kebutuhannya. Pengembang tampilan memiliki akses baca pada

hampir semua informasi yang tersedia pada Zotonic. Pada *template* dapat langsung memanggil kueri, mengambil *properties* dari pages, melakukan pengecekan kunci konfigurasi, dan lainnya. Sehingga *programmer* tidak butuh membuat *controller* lainnya atau mengadopsi beberapa *controller* untuk memberikan informasi ekstra kepada *template*. Dan pengembang tampilan tidak perlu untuk menunggu *programmer*. Hal ini berguna agar pengembang tampilan dapat secara langsung mengambil dan menampilkan data dari Zotonic pada *template*.

Selain itu, tampilan pada zotonic dibangun menggunakan *jQuery* dan CSS *framework Bootstrap* yang merupakan salah satu CSS *framework* yang sangat populer saat ini (Awwwards, 2017). Selain penggunaan *jQuery* dan *Bootstrap*, Zotonic juga mengadopsi penggunaan ErlyDTL (*Erlang implementation of the Django Template Language*). ErlyDTL digunakan agar pemuatan data dari basis data dapat dilakukan langsung dari *template* yang ingin memuat data itu sendiri. Zotonic sendiri menambahkan beberapa fitur pada ErlyDTL yang diadopsi seperti untuk *caching* dan *template* hanya di*compile* ke memori sehingga dapat mencegah masalah ketika logika internal dari *template* berubah versi.

Selain itu, zotonic menawarkan fitur fleksibilitas pada model data. Hal ini karena zotonic memberikan kebebasan kepada penggunanya untuk melakukan penambahan atau pengurangan terhadap model data. Model data pada Zotonic sendiri merupakan bentuk implementasi dari web semantik. Model datanya memiliki dua konsep utama yaitu *resource* dan *edge* (Zotonic, ndd). *Resource* pada Zotonic biasanya disebut sebagai *pages* pada halaman admin karena semua *resource* yang dihasilkan pada aplikasi akan ditampilkan sebagai sebuah *pages*. *Resources* memiliki bagian utama yaitu mereka memiliki *properties* seperti judul, ringkasan, dan isi bodi serta mereka milik dari sebuah *category*.



Gambar 2.4: Contoh Data Model
Sumber gambar: (Zotonic, ndd)

Pada Gambar 2.4, blok persegi menggambarkan *resource* dan panah menggambarkan *edge*. Alice adalah sebuah *resource* yang berkategori *person*. Begitu juga halnya dengan Bob yang merupakan *resource* dengan kategori *person*. Hal ini menunjukkan bahwa pada Zotonic, suatu kategori dapat memiliki banyak *instance* yang berupa *resource*. *Cooking* dan *Tutorials* sama-sama merupakan *resources* yang berkategori *keyword* sedangan *Cooking For Dummies* merupakan *resource* yang berkategori *article*. *has_keyword* merupakan *edge* yang menunjukan hubungan antara suatu *resource* dengan *resource* lainnya. Hal ini dapat dilihat bahwa *resource cooking for dummies* memiliki *keyword* yaitu *cooking* dan *tutorials*. Dengan kata lain, kategori *article* memiliki hubungan dengan kategori *keyword* yaitu *article* mempunyai *keyword*. Ini sama dengan relasi yang menghubungkan suatu *class* dengan *class* lainnya pada ontologi. Dengan model data yang seperti ini, zotonic merupakan salah satu CMS yang menerapkan implementasi dari web semantik.

2.3 *Software Product Line*

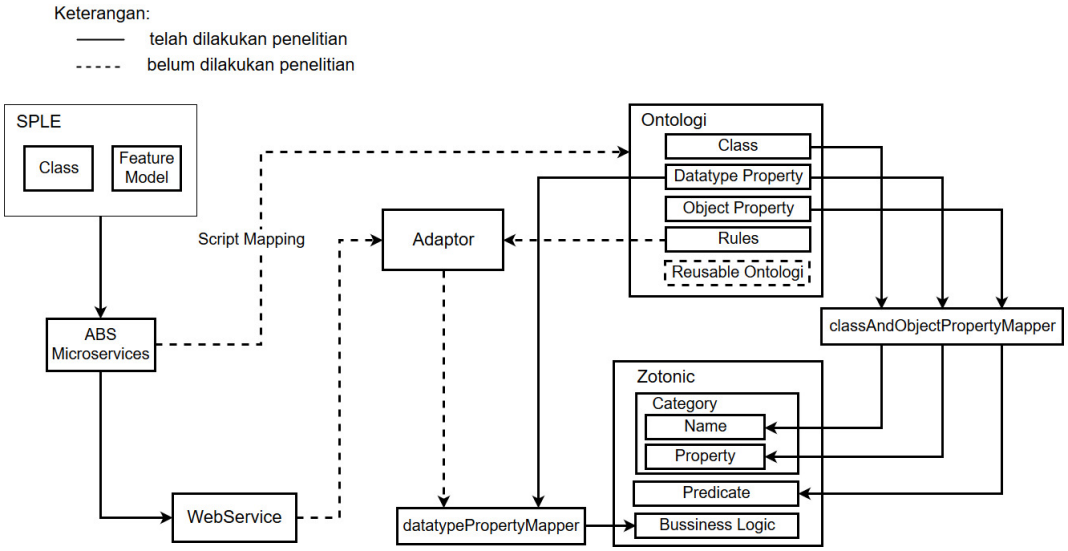
@todo
Jelasin SPL itu apa.

BAB 3
RANCANGAN

Dari permasalahan yang sudah didefinisikan, maka akan dibuat sebuah program yang akan melakukan integrasi ontologi dan *web service* dengan memanfaatkan Zotonic. Sebelum memulai pembuatan program, perlu dirancang bagaimana program ini akan berjalan. Pada bab ini, akan dibahas mengenai rancangan integrasi ontologi dan *web service* yang akan menggambarkan secara keseluruhan bagaimana program akan bekerja serta hal lainnya yang terhubungan dengan program dan juga mengenai rancangan adaptor yang akan menggambarkan secara dalam bagaimana program dapat mengakses *web service* dan terhubung dengan ontologi.

3.1 Rancangan Integrasi Ontologi dan *Web Service*

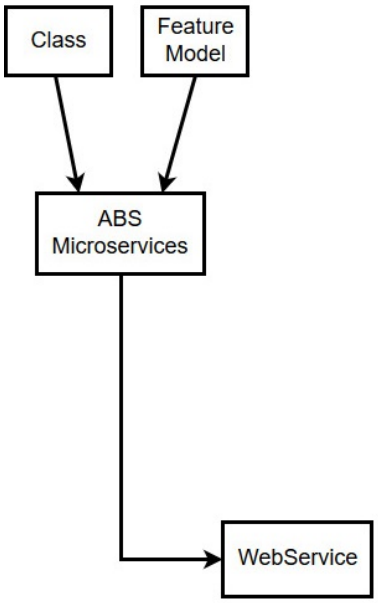
Agar dapat menghasilkan program yang bertujuan untuk melakukan integrasi ontologi dan *web services* maka perlu dirancang secara keseluruhan bagaimana program ini akan bekerja. Sesuai dengan kebutuhan dari program, maka integrasi ini akan dilakukan pada sebuah *framework* sekaligus *CMS* Zotonic, yang telah dimodifikasi agar dapat menerima masukan berupa ontologi. Secara garis besar, berikut merupakan gambaran cara program akan bekerja.



Gambar 3.1: Rancangan integrasi ontologi dan web service

Seperti yang dapat dilihat pada Gambar 3.1, *Class* dan *Feature Model* akan diproses menggunakan program translasi yang akan menghasilkan ABS *Microser-*

vices dimana ini telah dilakukan penelitian sebelumnya sehingga hal ini bukan merupakan bagian dari penelitian penulis. Setelah dihasilkan sebuah ABS *Microservices* dari proses translasi tersebut, maka ABS *Microservices* akan menghasilkan sebuah *web service* yang dapat digunakan oleh program lainnya. Hal ini dapat dilihat pada Gambar 3.2

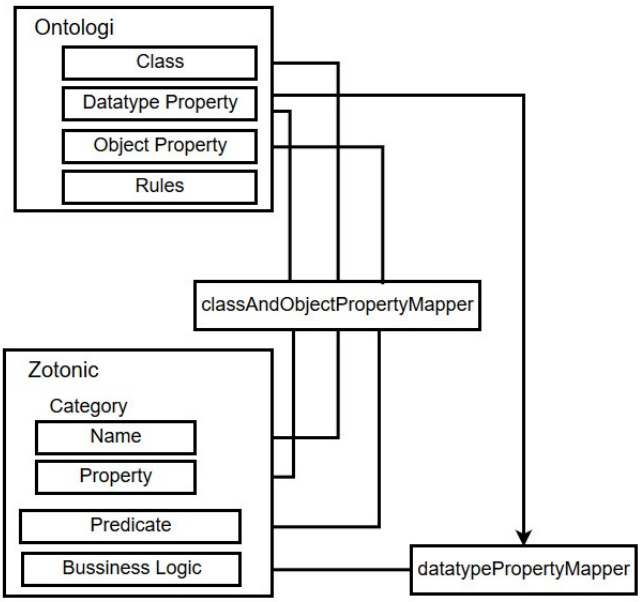


Gambar 3.2: Rancangan Bagian ABS

Dalam penelitian ini, *web services* yang dihasilkan oleh ABS *Microservices* akan digunakan oleh *Adaptor* yang akan terhubung dengan Zotonic. Selain digunakan untuk menghasilkan sebuah *web service*, ABS *Microservices* akan digunakan untuk menghasilkan sebuah ontologi menggunakan sebuah *script* yang akan melakukan pemetaan dari *class* dan *feature model* menjadi *class, datatype property, object property* dan *rules* pada ontologi. Untuk pemetaan ini sendiri tidak termasuk dalam penelitian ini karena penulis hanya akan menggunakan sebuah ontologi yang telah dirancang sebelumnya. Nantinya melalui proses pemetaan ini, akan dihasilkan sebuah ontologi yang bersifat *reusable*.

Menggunakan ontologi yang telah dirancang sebelumnya, ontologi tersebut akan dipetakan ke dalam struktur dari zotonic yang akan digunakan. Proses pemetaan dari ontologi ke dalam zotonic ini sendiri telah dilakukan pada penelitian sebelumnya oleh Bravyto dimana setiap *class* dari ontologi akan dipetakan menjadi nama dari kategori pada zotonic menggunakan *script classAndObjectPropertyMapper.sh*. Selain melakukan pemetaan pada *class, script* tersebut juga akan melakukan pemetaan *datatype property* pada ontologi menjadi *property* dari kategori pada zo-

tonic serta pemetaan *object property* pada ontologi menjadi *predicate* pada zotonic. Pada penelitian tersebut, dilakukan juga pemetaan dari *datatype property* menjadi *business logic* menggunakan *script* datatypePropertyMapper.sh. Proses ini digambarkan pada Gambar 3.3 berikut ini



Gambar 3.3: Rancangan Bagian Zotonic dan Ontologi

Namun pada penelitian tersebut, pembuatan *business logic* masih bersifat manual yang langsung ditaruh pada *script* datatypepropertyMapper.sh. Pada penelitian ini, penulis akan membuat sebuah adaptor yang akan memanggil *web service* sehingga *business logic* pada zotonic akan lebih fleksibel karena memanfaatkan *web service* serta memberikan kemudahan bagi *developer* dalam hal melakukan modifikasi.

3.2 Rancangan *Web Service*

Agar *web services* yang digunakan nantinya dapat sesuai dengan yang akan diimplementasikan, perlu dirancang terlebih dahulu bagaimana *web services* yang akan digunakan. Pertama perlu didefinisikan bagaimana struktur dari JSON yang diberikan diterima. Pada penelitian ini, penulis mendefinisikan struktur JSON sebagai berikut

Kode 3.1: Struktur JSON *web services*

```
{
  "status" : XXX,
  "message": "information about data",
  "data" : data
}
```

Pada kode 3.1, struktur JSON terdiri dari status,*message*, dan data. Status disini merupakan kode status yang didefinisikan agar memudahkan pada proses pembacaan hasil JSON nantinya. Berikut ini merupakan tabel mapping dari kode status

Tabel 3.1: Tabel Status Kode

Status	Keterangan
200	Mengembalikan data kepada pemanggil
201	Mengembalikan info sukses atau gagal
400	Masuk ke dalam *error exception*

Pada tabel 4.1, terdapat tabel status kode. Jika data hasil dari pemanggilan API tersebut perlu diketahui seperti untuk pemanggilan total donasi yang hasilnya perlu diketahui, maka status kode diatur menjadi 200. Jika pada API yang dilakukan seperti proses menambah data baru pada database, melakukan update pada database, serta melakukan penghapusan data pada database, tidak perlu dikembalikan datanya sehingga status kode diatur menjadi 201 dan *message* diatur menjadi informasi keberhasilan dari perintah yang dijalankan. Namun, jika proses *insert, update,* atau *delete* data pada database gagal dilakukan, atau data yang dicari pada database tidak ada maka status kode diset menjadi 400 dan *message* diset menjadi penyebab dari error tersebut.

Selain mendefinisikan status kode yang diberikan, perlu didefinisikan juga data apa yang akan dikirim seperti tabel berikut ini.

Tabel 3.2: Tabel data

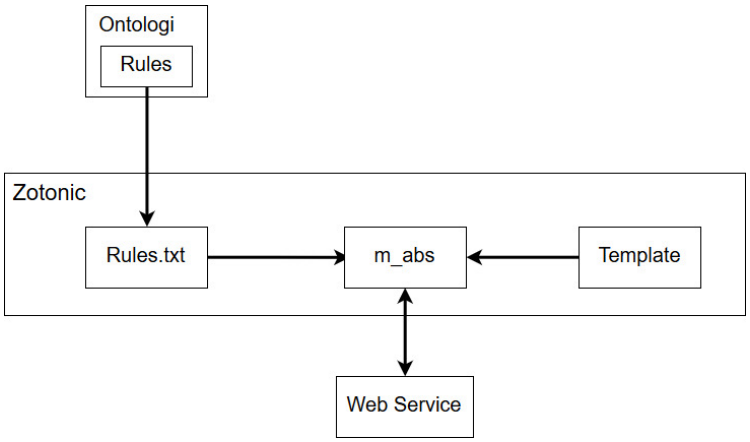
Status	Data
200	Data yang ingin dikirim
201	Data keseluruhan
400	null

Pada tabel 3.2, berdasarkan status kode yang telah didefinisikan pada tabel 4.1 maka jika status kode 200, data akan berisi data yang ingin dikirim. Sebagai contoh, jika total donasi berhasil dilakukan, maka status kode 200 dan data akan berisi

nominal dari total donasi. Untuk status kode 201, data akan berisi mengenai data yang diproses. Misalnya ingin menambahkan data pada database, dan jika berhasil maka status kode 201 dan data akan berisi seluruh data yang dimasukkan ke dalam database tersebut. Dan jika status kode 400, karena ini merupakan status kode untuk error maka data cukup diisi dengan null.

3.3 Rancangan *Adaptor*

Adaptor merupakan sebuah *script* yang akan memanggil *web service* sehingga dapat digunakan untuk melakukan pemrosesan *business logic* pada zotonic. Bagaimana adaptor akan bekerja sehingga menghasilkan business logic dapat dilihat pada gambar berikut ini.



Gambar 3.4: Rancangan Adaptor

Seperti yang dapat dilihat pada Gambar 3.4, *rules* yang terdapat pada ontologi akan dilakukan pemetaan menjadi tabel *rules* yang akan disimpan pada file rules.txt seperti yang terdapat pada kode 3.2. Namun, proses pemetaan ini berada diluar dari topik penelitian penulis sehingga untuk keperluan penelitian ini maka penulis membuat sebuah tabel *rules* secara manual untuk mengganti proses pemetaan tersebut.

Kode 3.2: Contoh tabel *rules*

```
{
  "createProgram": ["http://54.169.128.6:8080/abs/program/create"
     , 2],
  "createDonation": ["http://54.169.128.6:8080/abs/donation/
     create", 4],
  "updateProgram": ["http://54.169.128.6:8080/abs/program/update"
     , 2],
  "updateDonation": ["http://54.169.128.6:8080/abs/donation/
     update", 4],
  "deleteProgram": ["http://54.169.128.6:8080/abs/program/delete"
     , 1],
  "deleteDonation": ["http://54.169.128.6:8080/abs/donation/
     delete", 1],
  "totalDonation" : ["http://54.169.128.6:8080/abs/program/total-
     donation", 1]
}
```

Setelah terbentuk tabel *rules* pada file rules.txt, maka ketika model abs yang terdapat pada file dijalankan pada *template engine*, model abs akan membaca *rules* untuk mengetahui *endpoint* yang akan dijalankan pada proses pemanggilan *web service* serta untuk melakukan pengecekan apakah jumlah parameter yang dimasukkan telah sesuai dengan jumlah parameter yang diterima oleh *web service* atau tidak.

BAB 4
IMPLEMENTASI

Pada bab ini dijelaskan setiap hal yang dilakukan oleh penulis untuk melakukan implementasi terhadap rancangan yang telah dibuat pada bab sebelumnya. Penulis melakukan implementasi *adaptor* yang akan dipanggil melalui *template engine* dan juga melalui model lainnya. Selain itu, penulis juga melakukan *refactoring business logic* dengan memanfaatkan *adaptor* yang sudah diimplementasikan.

4.1 Interface *Adaptor*

Sebelum melakukan implementasi *adaptor*, perlu dibuat terlebih dahulu tabel *rules* yang akan dibaca oleh *adaptor*. Namun, karena belum adanya mekanisme pemetaan secara langsung dari *rules* yang dimiliki oleh ontologi ke dalam tabel *rules* maka penulis membuat tabel *rules* secara manual. Tabel *rules* tersebut berbentuk *json* dimana strukturnya sebagai berikut

Kode 4.1: Struktur tabel *rules*

```
{
nama fungsi : [endpoint, jumlah parameter]
}
```

Setelah tabel rules selesai dibuat sesuai dengan struktur pada kode 4.1, tabel rules tersebut disimpan ke dalam sebuah file yang bernama rules.txt dan ditaruh pada *root* dari zotonic. Untuk dapat menjalankan fungsi *adaptor* yang diinginkan, penulis membuat sebuah model baru pada zotonic dengan nama m_abs pada folder `root/src/models`. Untuk membuat sebuah model pada zotonic, zotonic mengharuskan setiap model untuk mengekspor beberapa fungsi yang dimiliki oleh zotonic yaitu m_find_value, m_to_list, dan m_value agar fungsi tersebut dapat digunakan melalui *template engine*.

Kode 4.2: Fungsi yang harus diekspor untuk model

```
-export([
m_find_value/3,
m_to_list/2,
m_value/2,
]).
```

Seperti yang dapat dilihat pada kode 4.2, agar fungsi tersebut dapat dijalankan pada *template engine*, tentu harus diimplementasi sesuai dengan kebutuhan. Sesuai dengan kebutuhannya agar *adaptor* dapat dipanggil melalui *template engine*, maka perlu didefinisikan terlebih dahulu bagaimana nantinya *adaptor* dipanggil. Pada penelitian ini, penulis mendefinisikan untuk pemanggilan *adaptor* pada *template engine* dilakukan dengan membuat perintah `m.abs.namaFungsi[{query param=value}]`. Implementasi dari fungsi m_find_value yang digunakan untuk melakukan pencocokan pola akan dijelaskan pada bagian berikutnya. Untuk fungsi m_to_list karena pada kasus ini tidak digunakan jadi cukup diimplementasikan seperti kode 4.3 berikut

Kode 4.3: Implementasi fungsi m_to_list

```
m_to_list(_, _Context) ->
[].
```

Sama seperti fungsi m_to_list, fungsi m_value juga tidak dibutuhkan pada implementasi *adaptor* sehingga dapat diimplementasikan seperti kode 4.4 berikut

Kode 4.4: Implementasi fungsi m_value

```
m_value(_, _Context) ->
undefined.
```

4.1.1 Pemanggilan *Adaptor* Melalui *Template Engine*

Setelah didefinisikan format pemanggilan *adaptor* pada *template engine*, maka akan diimplementasikan pencocokan pola pada proses pemanggilan *adaptor* dengan menggunakan fungsi m_find_value. Berikut adalah implementasi dari fungsi m_find_value

Kode 4.5: Implementasi fungsi m_find_value

```
% this method to handle call api from template
-spec m_find_value(Key, Source, Context) -> #m{} | undefined |
    any() when
Key:: integer() | atom() | string(),
Source:: #m{},
Context:: #context{}.

m_find_value(Type, #m{value=undefined} = M, _Context) ->
M#m{value=[Type]};

m_find_value({query, Query}, #m{value=Q} = _, _Context) when
    is_list(Q) ->
[Key] = Q,
[Url, Param] = lookup_rules(Key),
case validate_params(Param, Query) of
false ->
[{error, "Num of Params not same"}];
true ->
{DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode({
    Query})),
lager:info("ABS result : ~p", [DecodeJson]),
proplists:get_value(<<"data">>, DecodeJson)
end;

% Other values won't be processed
m_find_value(_, _, _Context) ->
undefined.
```

Pada kode 4.5, tahap awal untuk mengimplementasikan fungsi m_find_value adalah membuat sebuah *specifications* untuk fungsi tersebut. *Specifications* yang merupakan ketentuan yang harus dipenuhi mengenai *input* yang akan diterima oleh fungsi tersebut dan *output* yang akan dihasilkan oleh fungsi tersebut sehingga fungsi akan dijalankan jika dan hanya jika memenuhi dari *specifications* yang telah didefinisikan. Seperti yang sudah didefinisikan bahwa pemanggilan *adaptor* melalui *template engine* dengan cara `m.abs.namaFungsi[{query param=value}]`, maka ketika dijalankan m_abs akan menjalankan `m_find_value({query, Query}, #m{value=Q})` dimana namaFungsi yang didapatkan dari hasil pemanggilan pada *template engine* akan menjalankan fungsi `m_find_value(Type, #m{value=undefined})` untuk yang akan mengembalikan sebuah *maps* yang akan diterima kembali oleh fungsi `m_find_value({query, Query}, #m{value=Q})` yang menyimpan *maps* hasil kembalian tersebut ke dalam variabel Q. Lalu

parameter yang terdapat pada *template engine* akan disimpan oleh fungsi `m_find_value({query, Query}, #m{value=Q})` ke dalam variabel Query.

Setelah mendapatkan informasi tentang Query dan Q, fungsi m_find_value selanjutnya akan mengambil *key* yang akan dijalankan pada tabel *rules*. Langkah pertama, akan diekstrak *key* yang ingin dijalankan dari Q dengan cara *pattern matching*. Setelah mendapatkan *key* yang diinginkan, *key* tersebut akan dijadikan sebagai input untuk pemanggilan fungsi lookup_rules yang akan mengembalikan *endpoint* yang akan dipanggil oleh m_abs serta jumlah parameter dari fungsi yang akan dijalankan. Setelah mendapatkan *endpoint* serta jumlah parameter dari hasil pemanggilan fungsi lookup_rules yang berbentuk *list*, *list* tersebut akan diekstrak menjadi variabel Url yang menyimpan informasi *endpoint* dan variabel Param yang menyimpan informasi jumlah parameter dari fungsi tersebut. Selanjutnya fungsi m_find_value akan memanggil fungsi validate_params dengan parameter Param yang menyimpan informasi jumlah parameter dan variabel Query yang menyimpan informasi mengenai *list* dari parameter yang diberikan pada *template engine*.

Setelah mendapatkan hasil dari pemanggilan fungsi validate_params, pada fungsi m_find_value akan mengembalikan *error* jika hasil dari pemanggilan fungsi validate_params bernilai false. Sebaliknya, jika hasil dari pemanggilan fungsi validate_params bernilai true maka fungsi m_find_value akan memanggil fungsi fetch_data dengan parameter berupa variabel Url dan juga Parameter *Query* yang sudah dijadikan dalam bentuk *json* menggunakan `jiffy:encode/1`. Hasil dari pemanggilan fungsi fetch_data akan mengembalikan sebuah *json* yang berisikan data yang ingin ditampilkan. Sesuai dengan rancangan *web service* yang sudah dijelaskan pada bab sebelumnya, data yang ingin ditampilkan terdapat pada parameter "data" pada *json* sehingga kita perlu mengambil nilai dari parameter "data" tersebut dengan cara `proplists:get_value(«"data"», DecodeJson)` dimana `DecodeJson` merupakan *json* hasil dari fetch_data.

4.1.2 Pemanggilan *Adaptor* Melalui Model

Selain dipanggil melalui *template engine*, ada kebutuhan untuk memanggil *adaptor* dari model lain seperti pemanggilannya pada model m_rsc agar dapat mengirim data dari zotonic kepada *web service* melalui *adaptor*. Untuk itu, perlu diimplementasi sebuah fungsi baru dengan nama call_api_controller pada *adaptor* yang dapat diakses secara langsung oleh model lain seperti berikut

Kode 4.6: Implementasi fungsi untuk pemanggilan *adaptor* dari model

```
call_api_controller(Key, Data) ->
[Url, Param] = lookup_rules(Key),
case validate_params(Param, Data) of
false ->
[{error, "Num of Params not same"}];
true ->
lager:info("key ~p", [Data]),
lager:info("key ~s", [jiffy:encode({Data})]),
{DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode({
   Data})),
lager:info("[ABS] result ~p", [DecodeJson]),
case proplists:get_value(<<"status">>, DecodeJson) of
200 ->
DataResult = proplists:get_value(<<"data">>, DecodeJson),
lager:info("[ABS] status 200 ~p", [DataResult]);
201 ->
Message = proplists:get_value(<<"message">>, DecodeJson),
lager:info("[ABS] status 201 ~p", [binary_to_list(Message)]);
400 ->
Message = proplists:get_value(<<"message">>, DecodeJson),
lager:error("[ABS] status 400 ~p", [Message]);
_Other ->
lager:error("[ABS] status undefined ~p", [_Other])
end
end.
```

Pada kode 4.6, fungsi call_api_controller menerima dua parameter yaitu *Key* dan Data. *Key* merupakan nama fungsi yang ingin dipanggil oleh *web service*, dan Data merupakan data yang akan dikirim ke *web service*. Parameter *Key* yang didapatkan oleh fungsi call_api_controller akan digunakan untuk memanggil fungsi lookup_rules yang akan mengembalikan sebuah *list* yang berisi *endpoint* serta jumlah parameter dari fungsi yang akan dijalankan. *List* tersebut akan diekstrak dimana *endpoint* disimpan ke dalam variabel Url dan jumlah parameter disimpan ke dalam variabel Param. Setelah melakukan ekstraksi, fungsi call_api_controller akan memanggil fungsi validate_params dengan parameter Param yang menyimpan informasi jumlah parameter dan variabel Data yang menyimpan informasi mengenai *list* dari parameter yang ingin dikirim ke *web services*.

Setelah mendapatkan hasil dari pemanggilan fungsi validate_params, pada fungsi call_api_controller akan mengembalikan *error* jika hasil dari pemanggilan fungsi validate_params bernilai *false*. Sebaliknya, jika hasil dari pemanggilan fungsi validate_params bernilai *true* maka call_api_controller akan memanggil

fungsi fetch_data dengan parameter berupa variabel Url dan juga parameter data yang sudah dijadikan dalam bentuk *json* menggunakan *library* `jiffy:encode/1`. Hasil dari pemanggilan fungsi fetch_data akan mengembalikan sebuah *json* yang berisikan data yang ingin diperoleh. Setelah mendapatkan *json* dari pemanggilan fungsi fetch_data, dilakukan ekstraksi terhadap parameter "status" dari *json* tersebut. Tujuan dari proses ekstraksi tersebut agar diketahui data yang ingin diambil adalah parameter *"message"* atau parameter "data" dari *json* tersebut. Berikut merupakan tabel *mapping* yang digunakan setelah mendapatkan parameter "status" dari *json*

Tabel 4.1: Tabel *Mapping*

Status	Keterangan	Data yang diambil
200	Mengambil informasi	Data
201	Mengeksekusi perintah (*insert, update, delete*)	Message
400	*Error* dari API	Message
Other	*error*	Message

4.2 Implementasi *Adaptor*

4.2.1 Implementasi Fungsi *lookup_rules*

Fungsi lookup_rules merupakan fungsi yang membaca berkas rules.txt dan mengembalikan *list* yang berisikan *endpoint* dan jumlah parameter dari *key* yang diberikan sebagai *input*. Adapun implementasi dari fungsi lookup_rules sebagai berikut

Kode 4.7: Implementasi fungsi lookup_rules

```
lookup_rules(Key) ->
  File = ?RULES,
  case read_file(File) of
    {error, Error} ->
      [{error, Error}];
    [] ->
      [{error, "File empty"}];
    Json ->
      {DecodeJson} = jiffy:decode(Json),
      proplists:get_value(atom_to_binary(Key, latin1), DecodeJson
          )
end.

read_file(File) ->
  case file:read_file(File) of
    {ok, Data} ->
      Data;
    eof ->
      [];
    Error ->
      {error, Error}
end.
```

Pada kode 4.7, dapat dilihat bahwa fungsi lookup_rules akan menjalankan fungsi read_file yang membaca seluruh isi dari file yang telah didefinisikan sebagai tabel *rules*. Perlu didefinisikan juga dimana lokasi dari tabel *rules* yang akan dibaca sebagai variabel final dengan nama variabel *RULES* pada m_abs seperti kode 4.8

Kode 4.8: lokasi dari *file* rules.txt

```
-define(RULES, "./rules.txt").
```

4.2.2 Implementasi Fungsi *validate_params*

Implementasi dari fungsi validate_params dapat dilihat sebagai berikut

Kode 4.9: Implementasi fungsi validate_params

```
validate_params(Param, Query) ->
  case length(Query) == Param of
    false ->
      false;
    true ->
      true
end.
```

Pada kode 4.9, fungsi validate_params akan mengembalikan boolean hasil pengecekan jumlah parameter yang didapatkan dari tabel *rules* dan jumlah parameter yang terdapat pada *list* dari *Query*. Kembalikan *true* jika jumlah keduanya sama, dan kembalikan *false* jika jumlahnya tidak sama.

4.2.3 Implementasi Fungsi *fetch_data*

Adapun implementasi dari fungsi fetch_data sebagai berikut

Kode 4.10: Implementasi fungsi fetch_data

```
-spec fetch_data(Url, Query) -> list() when
  Url:: list(),
    Query:: list().
fetch_data(_,[]) ->
    [{error, "Params missing"}];
fetch_data("",_) ->
  [{error, "Url missing"}];
fetch_data(Url, Query) ->
      case post_page_body(Url, Query) of
          {error, Error} ->
              [{error, Error}];
          Json ->
              jiffy:decode(Json)
end.
```

4.3 Penggunaan *Adaptor* pada *Business Logic*

Pada penelitian sebelumnya, proses pembuatan *business logic* dilakukan secara langsung pada *script* datatypePropertyMapper.sh menggunakan *javascript*. Tetapi *business logic* antara satu situs dengan situs lainnya dapat berbeda sehingga dibu-

tuhkan sebuah mekanisme agar *business logic* dapat lebih dinamis berdasarkan ontologi yang digunakan. Untuk mengatasi hal tersebut, maka modul m_abs yang telah diimplementasi akan digunakan untuk membuat *business logic* agar bersifat dinamis. *Script* datatypePropertyMapper.sh akan menghasilkan *template* untuk admin dan juga *template* untuk halaman pengguna sehingga untuk menambahkan *business logic* yang bersifat dinamis dapat dimanfaatkan implementasi dari pemanggilan *adaptor* dari *template* yang sudah dijelaskan pada bagian sebelumnya. Berikut ini merupakan *business logic* untuk menghitung total sebelum dilakukan *refactoring*.

Kode 4.11: Fungsi total sebelum refactoring

```
{% javascript %}
  var predId = [];
  var total = 0;
{% endjavascript %}
{% with m.search.paged[{referrers id=id page=page}] as result
   %}
  {% for id, pred_id in result %}
    {% javascript %}
      if (predId.indexOf("{{ pred_id }}") == -1) {
        predId.push("{{ pred_id }}");
    {% endjavascript %}
    {% for amountholder in r.s[m.rsc[pred_id].title | lower] %}
      {% if amountholder.amount %}
        {% javascript %}
          total += {{ amountholder.amount }};
        {% endjavascript %}
      {% endif %}
    {% endfor %}
    {% javascript %}
      }
    {% endjavascript %}
  {% endfor %}
{% endwith %}
```

Pada kode 4.11, dapat dilihat bagaimana proses penjumlahan untuk fungsi total dilakukan secara *javascript*. Setelah dilakukan *refactoring* dengan cara memanfaatkan implementasi dari pemanggilan *adaptor* dari *template*, maka kode 4.11 akan diganti menjadi kode 4.12

Kode 4.12: Fungsi total setelah refactoring

```
{{m.abs.totalDonation[{query id=id}]}}
```

Pada kode 4.12, akan dipanggil *adaptor* dengan *key totalDonation* yang akan menjalankan fungsi untuk menghitung total donasi dengan id program yang sedang diakses baik pada *template* admin maupun *template* untuk pengguna.

BAB 5
HASIL

5.1 Perubahan pada Struktur Zotonic

Agar ontologi dan *web services* dapat terintegrasi, terdapat beberapa file baru yang ditambahkan ke dalam struktur zotonic. Berikut adalah struktur dari zotonic yang telah dapat membuat struktur zotonic dari ontologi sebelum penambahan file baru untuk menjalankan fungsi integrasi ontologi dan *web service*

Zotonic/
- .rebar/
- bin/
- deps/
- doc/
- docker/
- ebin/
- include/
- modules/
- priv/
- src/
- user/
- .dockerignore
- .editorconfig
- .travis.yml
- AUTHORS
- CONTRIBUTING.md
- CONTRIBUTORS
- Dockerfile
- Dockerfile.dev
- Dockerfile.heavy
- GNUmakefile
- LICENSE
- Makefile
- Readme.md

TRANSLATORS
USE_REBAR_LOCKED
build.cmd
charity_org_rdf.owl
classAndObjectPropertyMapper.sh
docker-compose.yml
prepare-release.sh
rebar
rebar.config
rebar.config.lock
rebar.config.lock.script
rebar.config.script
recentsite.txt
start.cmd
start.sh
zotonic.pid
zotonic_install

Setelah implementasi, perlu ditambahkan file rules.txt pada folder Zotonic dimana file ini merupakan tabel *rules* yang akan digunakan untuk *mapping web service* seperti yang dijelaskan pada bab sebelumnya. Selain itu, perlu ditambahkan juga modul m_abs.erl pada folder *models* yang terdapat pada folder *src* dari folder zotonic. modul ini yang nantinya akan berfungsi sebagai *adaptor* untuk menghubungkan antara ontologi dan *web services* seperti penjelasan pada bab sebelumnya.

5.2 Perubahan Setelah *Create Site Script* Dijalankan

Setelah menjalankan proses *build script*, maka *site* dapat dibuat dengan menjalankan proses *create site script*. Langkah untuk menjalankan *create site script* sebagai berikut

1. Jalankan zotonic dengan perintah seperti kode 5.1 untuk menjalankan zotonic dalam mode debug atau 5.2 untuk menjalankan zotonic tanpa debug.

Kode 5.1: Perintah untuk menjalankan Zotonic pada mode debug

```
$ bin/zotonic debug
```

Kode 5.2: Perintah untuk menjalankan Zotonic tanpa mode debug

```
$ bin/zotonic start
```

2. Persiapkan database pgsql untuk *site* yang akan dibuat dengan membuat *user* baru bernama zotonic dengan *password* zotonic pada pgsql. Lalu beri akses kepada *user* tersebut untuk dapat membuat database baru.

3. Setelah mempersiapkan database, selanjutnya edit file `/etc/hosts` dengan menambahkan namasite.dev yang diarahkan menuju local host seperti pada kode 5.3. namasite merupakan nama dari *site* yang ingin dibuat. Dalam penelitian ini, penulis membuat site bsmi sehingga namasite akan diubah menjadi bsmi

Kode 5.3: Configurasi file /etc/hosts

```
127.0.0.1   namasite.dev
```

4. Selanjutnya dapat menjalankan perintah seperti kode 5.4, untuk membuat *site* baru pada Zotonic dengan *template* blog

Kode 5.4: Perintah untuk membuat *site* baru pada Zotonic

```
$ bin/zotonic addsite -s blog namasite
```

5. Setelah proses *create site* selesai, berhentikan Zotonic dengan menekan `Ctrl + C` dua kali jika menggunakan mode debug, atau menggunakan perintah seperti kode 5.5 jika menjalankan zotonic tanpa mode debug

Kode 5.5: Perintah untuk memberhentikan zotonic

```
$ bin/zotonic stop
```

6. Selanjutnya jalankan perintah `make` untuk melakukan *build* ulang pada zotonic. Setelah selesai, jalankan lagi zotonic seperti pada langkah 1

Setelah proses diatas, zotonic akan membuat sebuah folder baru pada `user/site/` dengan nama folder sesuai dengan namasite yang dibuat. Pada penelitian ini, folder tersebut bernama bsmi karena namasite yang digunakan adalah bsmi. Perubahan yang terjadi dapat dilihat pada file page.tpl yang terletak pada folder *template* dari folder bsmi dimana untuk melakukan perhitungan *business logic* sudah memanfaatkan *adaptor* yang telah diimplementasikan pada bab sebelumnya.

Kode 5.6: *Business logic* untuk fungsi total pada kategori program

```
...
<p class="programtotal" id="programtotal">
  <b>{_ total _}</b> :
</p>
{% javascript %}
  var total = {{m.abs.totalDonation[{query id=id}]}};
  if (total == 0) {
    document.getElementById("programtotal").className += "
       hidden"; }
  else {
    document.getElementById("programtotal").innerHTML += total
       ;}
{% endjavascript %}
...
```

Pada kode 5.6 dapat dilihat bahwa variabel total akan menerima nilai dari hasil pemanggilan *adaptor* dengan fungsi *totalDonation*. Dengan memanfaatkan *adaptor*, akan mempermudah pengguna karena tidak perlu untuk melakukan implementasi *business logic* pada kode program.

5.3 Contoh Penerapan pada Web BSMI

BAB 6
PENUTUP

Pada bab terakhir ini,

6.1 Kesimpulan

6.2 Saran

DAFTAR REFERENSI

Awwwards (2017). What are frameworks? 22 best responsive css frameworks for web design. https://www.awwwards.com/what-are-frameworks-22-best-responsive-css-frameworks-for-web-design.html. [Diakses pada 28 Mei 2017].

Gruber, T. R. (1993). Toward principles for the design of ontologies used for knowledge sharing. pages 907–928. International Journal Human-Computer Studies.

Hopkins, M. dan Powell, J. (2015). *A Librarians Guide to Graphs, Data and the Semantic Web*. Elsevier Science.

Noy, N. F. dan McGuinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Medical Informatics.

OWL-Working-Group (2004). Owl web ontology language overview. https://www.w3.org/TR/2004/REC-owl-features-20040210/. [Diakses 24 Mei 2017].

Zotonic (n.d.a). Introduction - developer guide. http://docs.zotonic.com/en/latest/developer-guide/introduction.html. [Diakses 24 Mei 2017].

Zotonic (n.d.b). Mvc. http://zotonic.com/page/621/mvc. [Diakses pada 28 Mei 2017].

Zotonic (n.d.c). Speed. http://zotonic.com/page/614/speed. [Diakses pada 28 Mei 2017].

Zotonic (n.d.d). The zotonic data model. http://docs.zotonic.com/en/latest/user-guide/data-model.html. [Diakses pada 28 Mei 2017].

LAMPIRAN

LAMPIRAN 1 : KODE SUMBER MODEL ABS

m_abs.erl

Skrip ini diletakkan pada direktori `/usr/sesuatu` dan hanya dapat dieksekusi oleh *root*. Skrip ini berguna untuk menambahkan pengguna baru sesuai dengan konfigurasi baru yang telah ditetapkan.

Kode 1: Skrip adaptor m_abs.erl

```
%% @author Andri Kurniawan <andrikurniawan.id@gmail.com>
%% @copyright 2017 Andri Kurniawan
%% Date: 2017-05-11
%%
%% @doc Template access for abs model

%% Copyright 2017 Andri Kurniawan
%%
%% Licensed under the Apache License, Version 2.0 (the "License");
%% you may not use this file except in compliance with the License.
%% You may obtain a copy of the License at
%%
%%     http://www.apache.org/licenses/LICENSE-2.0
%%
%% Unless required by applicable law or agreed to in writing, software
%% distributed under the License is distributed on an "AS IS" BASIS,
%% WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
%% See the License for the specific language governing permissions and
%% limitations under the License.
-module(m_abs).
-behaviour(gen_model).

-export([
  m_find_value/3,
  m_to_list/2,
  m_value/2,
  call_api_controller/2
]).

-include_lib("zotonic.hrl").

-define(RULES, "/home/andri/skripsi/zotonic/rules.txt").

% this method to handle call api from template
-spec m_find_value(Key, Source, Context) -> #m{} | undefined | any() when
    Key:: integer() | atom() | string(),
    Source:: #m{},
    Context:: #context{}.

m_find_value(Type, #m{value=undefined} = M, _Context) ->
    M#m{value=[Type]};
```

```
m_find_value({query, Query}, #m{value=Q} = _, _Context) when is_list(Q) ->
  [Key] = Q,
  [Url, Param] = lookup_rules(Key),
  case validate_params(Param, Query) of
    false ->
      [{error, "Num of Params not same"}];
    true ->
      {DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode({Query})),
      lager:info("ABS result : ~p", [DecodeJson]),
      proplists:get_value(<<"data">>, DecodeJson)
  end;

% Other values won't be processed
m_find_value(_, _, _Context) ->
    undefined.

m_to_list(_, _Context) ->
  [].

m_value(_, _Context) ->
  undefined.

% this method to handle call api from another module
call_api_controller(Key, Data) ->
  [Url, Param] = lookup_rules(Key),
  case validate_params(Param, Data) of
    false ->
      [{error, "Num of Params not same"}];
    true ->
      lager:info("key ~p", [Data]),
      lager:info("key ~s", [jiffy:encode({Data})]),
      {DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode({Data})),
      lager:info("[ABS] result ~p", [DecodeJson]),
      case proplists:get_value(<<"status">>, DecodeJson) of
        200 ->
          DataResult = proplists:get_value(<<"data">>, DecodeJson),
          lager:info("[ABS] status 200 ~p", [DataResult]);
        201 ->
          Message = proplists:get_value(<<"message">>, DecodeJson),
          lager:info("[ABS] status 201 ~p", [binary_to_list(Message)]);
        400 ->
          Message = proplists:get_value(<<"message">>, DecodeJson),
          lager:error("[ABS] status 400 ~p", [Message]);
        _Other ->
          lager:error("[ABS] status undefined ~p", [_Other])
      end
  end.

-spec fetch_data(Url, Query) -> list() when
  Url:: list(),
    Query:: list().
fetch_data(_,[]) ->
    [{error, "Params missing"}];
fetch_data("",_) ->
  [{error, "Url missing"}];
fetch_data(Url, Query) ->
```

```
      case post_page_body(Url, Query) of
          {error, Error} ->
              [{error, Error}];
          Json ->
              jiffy:decode(Json)
      end.

post_page_body(Url, Body) ->
  case httpc:request(post, {Url, [], "application/json", Body}, [], []) of
    {ok,{_, _, Response}} ->
      Response;
    Error ->
      {error, Error}
  end.

lookup_rules(Key) ->
  File = ?RULES,
  case read_file(File) of
    {error, Error} ->
      [{error, Error}];
    [] ->
      [{error, "File empty"}];
    Json ->
      {DecodeJson} = jiffy:decode(Json),
      proplists:get_value(atom_to_binary(Key, latin1), DecodeJson)
  end.

read_file(File) ->
  case file:read_file(File) of
    {ok, Data} ->
      Data;
    eof ->
      [];
    Error ->
      {error, Error}
  end.

validate_params(Param, Query) ->
  case length(Query) == Param of
    false ->
      false;
    true ->
      true
end.
```

LAMPIRAN 2 : KODE SUMBER RULES

rules.txt

Kode 2: Berkas `compute.xml`

```
{
  "createProgram": ["http://54.169.128.6:8080/abs/program/create"
     , 2],
  "createDonation": ["http://54.169.128.6:8080/abs/donation/
     create", 4],
  "updateProgram": ["http://54.169.128.6:8080/abs/program/update"
     , 2],
  "updateDonation": ["http://54.169.128.6:8080/abs/donation/
     update", 4],
  "deleteProgram": ["http://54.169.128.6:8080/abs/program/delete"
     , 1],
  "deleteDonation": ["http://54.169.128.6:8080/abs/donation/
     delete", 1],
  "totalDonation" : ["http://54.169.128.6:8080/abs/program/total-
     donation", 1]
}
```

LAMPIRAN 3 : KODE SUMBER *WEB SERVICE*

DonationController.java

Kode 3: Berkas DonationController.java

```
package com.skripsi.Controller;

import com.skripsi.Domain.DonationEntity;
import com.skripsi.Domain.Wrapper;
import com.skripsi.Service.DonationService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/

@RestController
@RequestMapping("/donation")
public class DonationController {

  @Autowired
  private DonationService donationService;

  @RequestMapping(path = "/create", method = RequestMethod.POST)
  public Wrapper create(@RequestBody DonationEntity
      donationEntity){
    DonationEntity donation = new DonationEntity();
    donation.setId(donationEntity.getId());
    donation.setName(donationEntity.getName());
    donation.setAmount(donationEntity.getAmount());
    donation.setpId(donationEntity.getpId());
    DonationEntity hasil = donationService.save(donation);

    if(hasil == null){
      return new Wrapper(400, "Gagal menyimpan donasi baru", null
          );
    }
    return new Wrapper(201, "Donasi berhasil disimpan", donation)
        ;
```

```
  }

  @RequestMapping(path = "/update", method = RequestMethod.POST)
  public Wrapper update(@RequestBody DonationEntity
      donationEntity) {
    DonationEntity donation = donationService.findById(
        donationEntity.getId());
    donation.setName(donationEntity.getName());
    donation.setAmount(donationEntity.getAmount());
    donation.setpId(donationEntity.getpId());
    DonationEntity hasil = donationService.save(donationEntity);

    if(hasil == null){
      return new Wrapper(400, "Gagal memperbaharui donasi", null)
          ;
    }
    return new Wrapper(201, "Donasi berhasil diperbaharui",
        donation);
  }

  @RequestMapping(path = "/delete", method = RequestMethod.POST)
  public Wrapper delete(@RequestBody DonationEntity
      donationEntity){
    int result = donationService.delete(donationEntity.getId());
    if (result == 0){
      return new Wrapper(400, "Gagal menghapus program", null);
    }
    return new Wrapper(201, "Berhasil menghapus program", result)
        ;
  }
}
```

ProgramController.java

Kode 4: Berkas ProgramController.java

```
package com.skripsi.Controller;

import com.skripsi.Domain.ProgramEntity;
import com.skripsi.Domain.Wrapper;
import com.skripsi.Service.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/

@RestController
@RequestMapping("/program")
public class ProgramController {

  @Autowired
  private ProgramService programService;

  @Autowired
  private DonationService donationService;

  @RequestMapping(path = "/create", method = RequestMethod.POST)
  public Wrapper create(@RequestBody ProgramEntity programEntity)
      {
    ProgramEntity program = new ProgramEntity();
    program.setId(programEntity.getId());
    program.setName(programEntity.getName());
    ProgramEntity hasil = programService.save(program);

    if(hasil == null){
      return new Wrapper(400, "Gagal menyimpan program baru",
          null);
    }
    return new Wrapper(201, "Program berhasil disimpan", program)
        ;
  }
```

```
  @RequestMapping(path = "/update", method = RequestMethod.POST)
  public Wrapper update(@RequestBody ProgramEntity programEntity)
      {
    ProgramEntity program = programService.findById(programEntity
        .getId());
    program.setName(programEntity.getName());
    ProgramEntity hasil = programService.save(program);

    if(hasil == null){
      return new Wrapper(400, "Gagal memperbaharui program", null
          );
    }
    return new Wrapper(201, "Program berhasil diperbaharui",
        program);
  }

  @RequestMapping(path = "/delete", method = RequestMethod.POST)
  public Wrapper delete(@RequestBody ProgramEntity programEntity)
      {
    int result = programService.delete(programEntity.getId());
    if (result == 0){
      return new Wrapper(400, "Gagal menghapus program", null);
    }
    return new Wrapper(201, "Berhasil menghapus program", result)
        ;
  }

  @RequestMapping(path = "/total-donation", method =
      RequestMethod.POST)
  public Wrapper totalDonation(@RequestBody ProgramEntity
      programEntity){
    int total = donationService.getTotalDonationOfProgram(
        programEntity.getId());
    if(total < 1){
      return new Wrapper(400, "Tidak ada donasi untuk program ini
          ", 0);
    }
    return new Wrapper(200, "Berhasil", total);
  }

}
```

DonationEntity.java

Kode 5: Berkas DonationEntity.java

```
package com.skripsi.Domain;

import javax.persistence.*;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/
@Entity
@Table(name = "donation", schema = "abs_program")
public class DonationEntity {
  private int id;
  private String name;
  private int amount;
  private int pId;

  @Id
  @Column(name = "id")
  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  @Basic
  @Column(name = "name")
  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  @Basic
  @Column(name = "amount")
  public int getAmount() {
    return amount;
  }
```

```
  public void setAmount(int amount) {
    this.amount = amount;
  }

  @Basic
  @Column(name = "p_id")
  public int getpId() {
    return pId;
  }

  public void setpId(int pId) {
    this.pId = pId;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    DonationEntity that = (DonationEntity) o;

    if (id != that.id) return false;
    if (amount != that.amount) return false;
    if (pId != that.pId) return false;
    if (name != null ? !name.equals(that.name) : that.name !=
        null) return false;

    return true;
  }

  @Override
  public int hashCode() {
    int result = id;
    result = 31 * result + (name != null ? name.hashCode() : 0);
    result = 31 * result + amount;
    result = 31 * result + pId;
    return result;
  }
}
```

ProgramEntity.java

Kode 6: Berkas ProgramEntity.java

```
package com.skripsi.Domain;

import javax.persistence.*;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/
@Entity
@Table(name = "program", schema = "abs_program", catalog = "")
  public class ProgramEntity {
  private int id;
  private String name;

  @Id
  @Column(name = "id")
  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  @Basic
  @Column(name = "name")
  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    ProgramEntity that = (ProgramEntity) o;
```

```
    if (id != that.id) return false;
    if (name != null ? !name.equals(that.name) : that.name !=
        null) return false;

    return true;
  }

  @Override
  public int hashCode() {
    int result = id;
    result = 31 * result + (name != null ? name.hashCode() : 0);
    return result;
  }
}
```

Wrapper.java

Kode 7: Berkas Wrapper.java

```
package com.skripsi.Domain;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/
public class Wrapper <E> {
  private int status;
  private String message;
  private E data;

  /**
  * Returns the status of this wrapper.
  * @return the {@code status} property
  */
  public int getStatus() {
    return status;
  }

  /**
  * Sets the status of this wrapper with the specified {@code
     status}.
  * @param status the {@code status} for this wrapper
  */
  public void setStatus(int status) {
    this.status = status;
  }

  /**
  * Returns the message of this wrapper.
  * @return the {@code message} property
  */
  public String getMessage() {
    return message;
  }

  /**
  * Sets the message of this wrapper with the specified {@code
     message}.
  * @param message the {@code message} for this wrapper
  */
```

```
  public void setMessage(String message) { this.message = message
      ; }

  /**
  * Returns the aata of this wrapper.
  * @return the {@code data} property
  */
  public E getData() { return data; }

  /**
  * Sets the data of this wrapper with the specified {@code data
      }.
  * @param data the {@code data} for this wrapper
  */
  public void setData(E data) { this.data = data; }

  /**
  * Constructs a new Wrapper with the specified status, message,
      and data.
  * @param status the status for this wrapper
  * @param message the message for this wrapper
  * @param data the data for this wrapper
  */
  public Wrapper(int status, String message, E data) {

    this.status = status;
    this.message = message;
    this.data = data;
  }
}
```

DonationRepository.java

Kode 8: Berkas DonationRepository.java

```
package com.skripsi.Repository;

import com.skripsi.Domain.DonationEntity;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import javax.transaction.Transactional;
import java.util.List;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/
@Repository
public interface DonationRepository extends CrudRepository<
    DonationEntity, Integer> {
  DonationEntity findById(int id);
  List<DonationEntity> findBypId(int p_id);

  @Transactional
  Integer deleteById(int id);

  @Query(value = "Select sum(amount) from donation where p_id = :
      p_id",
  nativeQuery = true)
  Integer getTotalDonationOfProgram(@Param("p_id") int p_id);
}
```

ProgramRepository.java

Kode 9: Berkas ProgramRepository.java

```
package com.skripsi.Repository;

import com.skripsi.Domain.ProgramEntity;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import javax.transaction.Transactional;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/
@Repository
public interface ProgramRepository extends CrudRepository<
    ProgramEntity, Integer> {
  ProgramEntity findById(int id);
  @Transactional
  Integer deleteById(int id);
}
```

DonationService.java

Kode 10: Berkas DonationService.java

```
package com.skripsi.Service;

import com.skripsi.Domain.DonationEntity;
import com.skripsi.Repository.DonationRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/

@Service
public class DonationService {
  @Autowired
  private DonationRepository donationRepository;

  public DonationEntity findById(int id){
    return donationRepository.findById(id);
  }

  public int getTotalDonationOfProgram(int p_id){
    if (donationRepository.getTotalDonationOfProgram(p_id) ==
        null)
      return 0;
    else
      return donationRepository.getTotalDonationOfProgram(p_id);
  }

  public List<DonationEntity> findByIdProgram(int p_id){
    return donationRepository.findBypId(p_id);
  }

  public DonationEntity save(DonationEntity donation){
    return donationRepository.save(donation);
  }

  public int delete(int id){
    return donationRepository.deleteById(id);
```

```
    }
}
```

ProgramService.java

Kode 11: Berkas ProgramService.java

```
package com.skripsi.Service;

import com.skripsi.Domain.ProgramEntity;
import com.skripsi.Repository.ProgramRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
* Created by Andri Kurniawan <andrikurniawan.id@gmail.com>
* on 5/13/2017.
*/

@Service
public class ProgramService {
  @Autowired
  private ProgramRepository programRepository;

  public ProgramEntity findById(int id){
    return programRepository.findById(id);
  }

  public ProgramEntity save(ProgramEntity program){
    return programRepository.save(program);
  }

  public int delete(int id){
    return programRepository.deleteById(id);
  }
}
```