



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 kehadiran 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 penggerjaan 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 penggerjaan skripsi ini.
2. Bapak Drs. Lim Yohanes Stefanus M.Math., Ph.D selaku dosen pembimbing akademis penulis yang selalu membimbing, memberi masukan dan 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 dan menjadi teman untuk berdiskusi terkait hal teknis selama penggerjaan skripsi ini.
5. Bu Maya, Kak Iis, Kak Niken serta teman-teman Lab RSE lainnya yang turut memberikan masukan dan bantuan kepada penulis.
6. Nabilah Akiti Hara selaku pacar dari penulis yang selalu memberikan motivasi dan mendukung penulis selama penggerjaan skripsi serta menemani penulis melalui aplikasi *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 jemu 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 Non-eksklusif 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

Pembuatan web semantik yang belum umum dan susah, menjadikan web semantik belum populer saat ini. Padahal dengan menggunakan web semantik, informasi-informasi yang ada pada web dapat diolah secara langsung oleh komputer. Penelitian terus dilakukan untuk memudahkan pembuatan web semantik dimana salah satunya adalah Zotonic. Zotonic merupakan salah satu web *framework* yang berbasis semantik. Pada penelitian sebelumnya telah dikembangkan Zotonic yang dapat menerima ontologi sebagai masukan untuk pembentukan struktur webnya. Namun, karena ontologi tidak mengandung *business logic*, maka diperlukan mekanisme untuk menghubungkan ontologi dengan ABS *microservices* agar *business logic* pada Zotonic dapat bersifat dinamis berdasarkan kebutuhan. Penelitian ini membahas bagaimana ontologi dan *web service* yang dihasilkan oleh ABS *microservices* dapat diintegrasikan. Hal ini berguna agar dapat menciptakan beberapa web pada Zotonic yang memiliki struktur yang sama namun memiliki *business logic* yang berbeda. Tahapan dari penelitian ini adalah melakukan rancangan terhadap integrasi yang akan dilakukan dan implementasi *adaptor* yang digunakan sebagai penghubung antara ontologi dan *web services*. Pada akhir penelitian ini, dilakukan ujicoba untuk melihat hasil dari implementasi yang telah dilakukan.

Kata Kunci:

ABS, *Adaptor*, Ontologi, SPL, *Web Service*, Zotonic

ABSTRACT

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

Making semantic web is not yet common and difficult, making semantic web not yet popular at this time. Whereas, by using semantic web, the information available on the web can be processed directly by computer. Research continues to be done to facilitate the manufacture of semantic web where one of them is Zotonic. Zotonic is one of the semantic-based web frameworks. In the previous research has been developed Zotonic that can accept ontology as input for the formation of web structure. However, since ontology does not contain business logic, a mechanism for connecting ontology with ABS microservices is required so that business logic on Zotonic can be dynamic based on need. This study discusses how the ontology and web services produced by ABS microservices can be integrated. This is useful in order to create multiple webs on Zotonic that have the same structure but have different business logic. The stages of this research is to design the integration that will be done and the implementation of the adaptor used as a liaison between ontology and web services. At the end of this study, a trial is conducted to see the results of the implementation that has been done.

Keywords:

ABS, Adaptor, Ontology, SPL, Web Service, Zotonic

DAFTAR ISI

HALAMAN JUDUL	i
LEMBAR PERNYATAAN ORISINALITAS	ii
LEMBAR PENGESAHAN	iii
KATA PENGANTAR	iv
LEMBAR PERSETUJUAN PUBLIKASI ILMIAH	vi
ABSTRAK	vii
Daftar Isi	ix
Daftar Gambar	x
Daftar Tabel	xi
Daftar Kode	xii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Perumusan Masalah	3
1.3 Tujuan Penelitian	3
1.4 Ruang Lingkup Penelitian	3
1.5 Sistematika Penulisan	3
2 TINJAUAN PUSTAKA	5
2.1 Ontologi	5
2.2 Web Framework Berbasis Semantic: Zotonic	8
2.3 Software Product Line	10
3 RANCANGAN	12
3.1 Rancangan Integrasi Ontologi dan Web Service	12
3.2 Rancangan Web Service	14
3.3 Rancangan Adaptor	16
4 IMPLEMENTASI	18
4.1 Interface Adaptor	18
4.1.1 Pemanggilan Adaptor Melalui Template Engine	19
4.1.2 Pemanggilan Adaptor Melalui Model	21
4.2 Implementasi Adaptor	23
4.2.1 Implementasi Fungsi <i>lookup_rules</i>	23

4.2.2	Implementasi Fungsi <i>validate_params</i>	25
4.2.3	Implementasi Fungsi <i>fetch_data</i>	25
4.3	Penggunaan <i>Adaptor</i> pada <i>Business Logic</i>	26
5	HASIL	28
5.1	Perubahan pada Struktur Zotonic	28
5.2	Perubahan Setelah <i>Create Site Script</i> Dijalankan	29
5.3	Contoh Penerapan pada Web BSMI	31
5.4	Contoh Keterhubungan Data pada Web BSMI	48
5.5	Kesesuaian Keterhubungan Data dengan Ontologi	51
6	PENUTUP	53
6.1	Kesimpulan	53
6.2	Saran	54
Daftar Referensi		55
LAMPIRAN		1
Lampiran 1 : Kode Sumber Model ABS		2
Lampiran 2 : Kode Sumber rules		4
Lampiran 3 : Struktur Zotonic		5
Lampiran 4 : Kode Sumber <i>Web Service</i>		7

DAFTAR GAMBAR

1.1	<i>Roadmap</i> pembentukan <i>web</i> berbasis semantik	2
2.1	Contoh Kelas pada Ontologi	6
2.2	Contoh Relasi pada Ontologi	6
2.3	Contoh Struktur Hierarki pada <i>Charity Organization</i>	7
2.4	Contoh Data Model	9
2.5	Proses pembuatan produk menggunakan metode SPL	11
3.1	Rancangan integrasi ontologi dan <i>web service</i>	12
3.2	Rancangan Bagian ABS	13
3.3	Rancangan Bagian Zotonic dan Ontologi	14
3.4	Rancangan <i>Adaptor</i>	16
5.1	Tampilan awal <i>site</i> pada Zotonic	31
5.2	Tampilan login admin pada Zotonic	32
5.3	Tampilan <i>dashboard</i> admin pada Zotonic	33
5.4	Tampilan membuat <i>page</i> pada Zotonic	33
5.5	Tampilan halaman mengubah detail <i>page</i> kategori program	34
5.6	Tampilan halaman mengubah detail <i>page</i> kategori program	34
5.7	Penyimpanan <i>page</i> program pada database eksternal	35
5.8	Tampilan <i>page</i> Pelantikan dan Seminar BSMI Brebes	35
5.9	Tampilan membuat <i>page</i> pada Zotonic	36
5.10	Tampilan halaman untuk mengubah donasi pada Zotonic	37
5.11	Tampilan halaman mengubah detail <i>page</i> kategori donasi	37
5.12	Tampilan halaman mengubah detail <i>page</i> kategori donasi	38
5.13	Tampilan halaman mengubah detail <i>page</i> kategori donasi	38
5.14	Tampilan halaman mengubah detail <i>page</i> kategori donasi	39
5.15	Penyimpanan donasi pada database eksternal	39
5.16	Tampilan <i>page</i> Pelantikan dan Seminar BSMI Brebes setelah pembuatan donasi	40
5.17	Tampilan membuat <i>page</i> pada Zotonic	41
5.18	Tampilan halaman mengubah detail <i>page</i> pada kategori Donasi	41
5.19	Tampilan halaman mengubah detail <i>page</i> pada kategori Donasi	42
5.20	Tampilan halaman mengubah detail <i>page</i> pada kategori Donasi	42
5.21	Tampilan halaman mengubah detail <i>page</i> pada kategori Donasi	43
5.22	Penyimpanan donasi pada database eksternal	43
5.23	Tampilan <i>page</i> Pelantikan dan Seminar BSMI Brebes setelah penambahan donasi	44
5.24	Tampilan halaman mengubah detail <i>page</i> kategori program setelah ada donasi	44
5.25	Tampilan halaman mengubah detail <i>page</i> pada kategori Donasi	45

5.26 Tampilan konfirmasi penghapusan <i>page</i>	45
5.27 Penghapusan tupel pada database eksternal setelah penghapusan donasi	46
5.28 Tampilan <i>page</i> Pelantikan dan Seminar BSMI Brebes setelah penghapusan donasi	47
5.29 Tampilan halaman mengubah detail <i>page</i> pada kategori Donasi	47
5.30 Perubahan database eksternal setelah perubahan donasi	48
5.31 Tampilan <i>page</i> Pelantikan dan Seminar BSMI Brebes setelah perubahan	48
5.32 Perbedaan <i>web</i> semantik dengan <i>web</i> biasa	49
5.33 Tampilan <i>page</i> <i>Program</i> yang telah menampilkan keterhubungan data	50
5.34 Tampilan <i>page</i> <i>Donasi</i> yang telah menampilkan keterhubungan data	50
5.35 Tampilan <i>page</i> <i>Donasi</i> yang telah menampilkan keterhubungan data	51
5.36 Potongan graf dari ontologi <i>Charity Organization</i>	51

DAFTAR TABEL

3.1	Tabel Status Kode	15
3.2	Tabel data	15
4.1	Tabel <i>Mapping</i>	23
5.1	Perbandingan struktur Zotonic yang asli dan hasil penelitian sebelumnya	28

DAFTAR KODE

3.1	Struktur JSON <i>web services</i>	15
3.2	Contoh tabel <i>rules</i>	17
4.1	Fungsi yang harus diekspor untuk <i>model</i>	18
4.2	Implementasi fungsi <i>m_to_list</i>	19
4.3	Implementasi fungsi <i>m_value</i>	19
4.4	Implementasi fungsi <i>m_find_value</i>	20
4.5	Implementasi fungsi untuk pemanggilan <i>adaptor</i> dari <i>model</i>	22
4.6	Struktur tabel <i>rules</i>	23
4.7	Implementasi fungsi <i>lookup_rules</i>	24
4.8	lokasi dari file <i>rules.txt</i>	24
4.9	Implementasi fungsi <i>validate_params</i>	25
4.10	Implementasi fungsi <i>fetch_data</i>	25
4.11	Implementasi fungsi <i>post_page_body</i>	26
4.12	Fungsi total sebelum <i>refactoring</i>	27
4.13	Fungsi total setelah refactoring	27
5.1	Perintah untuk menjalankan Zotonic pada mode <i>debug</i>	29
5.2	Perintah untuk menjalankan Zotonic tanpa mode <i>debug</i>	30
5.3	Configurasi berkas /etc/hosts	30
5.4	Perintah untuk membuat <i>site</i> baru pada Zotonic	30
5.5	Perintah untuk memberhentikan Zotonic	30
5.6	<i>Business logic</i> untuk fungsi total pada kategori program	31
1	Berkas adaptor <i>m_abs.erl</i>	2
2	Berkas <i>rules.txt</i>	5
3	Berkas <i>DonationController.java</i>	8
4	Berkas <i>ProgramController.java</i>	10
5	Berkas <i>DonationEntity.java</i>	12
6	Berkas <i>ProgramEntity.java</i>	14
7	Berkas <i>Wrapper.java</i>	16
8	Berkas <i>DonationRepository.java</i>	18
9	Berkas <i>ProgramRepository.java</i>	19
10	Berkas <i>DonationService.java</i>	20
11	Berkas <i>ProgramService.java</i>	22

BAB 1

PENDAHULUAN

Bab ini menjelaskan latar belakang permasalahan, perumusan masalah, tujuan penelitian ini dilakukan, ruang lingkup penelitian serta sistematika penulisan.

1.1 Latar Belakang

Penggunaan *web* sebagai sarana untuk berbagi informasi pada saat ini sangat tinggi. Namun, tidak banyak orang yang mampu mengembangkan *web* serta tidak adanya anggaran untuk pembuatan *web* menjadi masalah tersendiri terutama bagi organisasi non-profit. Selain itu, banyaknya informasi yang tersebar di *web* saat ini hanya sebatas informasi saja yang tidak dapat diolah. Hal ini karena informasi-informasi tersebut tidak memiliki hubungan yang terstruktur dan hanya didesain untuk manusia saja sehingga program komputer tidak dapat mengolah informasi tersebut (Berners-Lee et al., 2001).

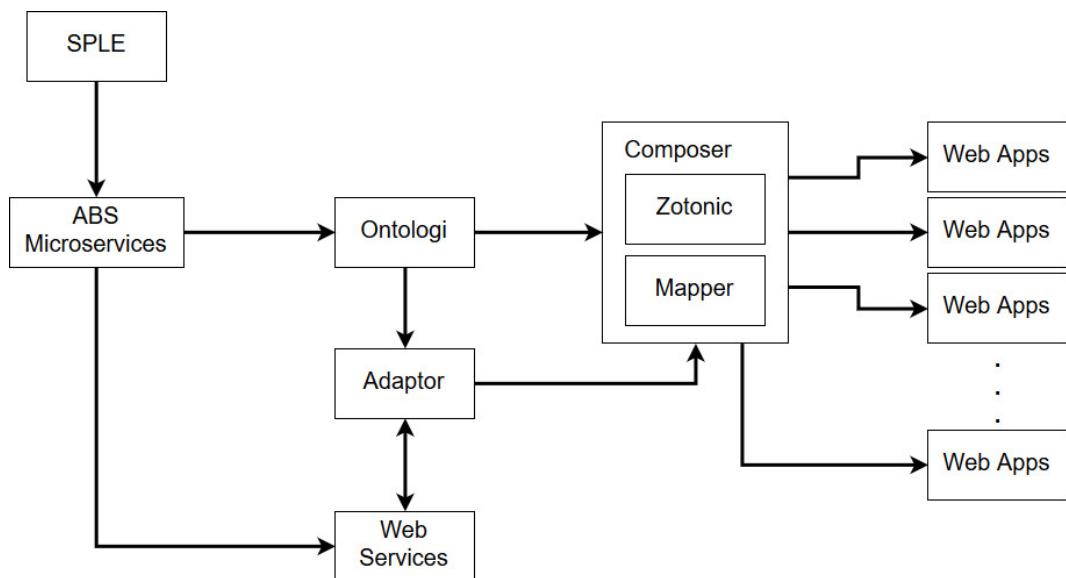
Selain permasalahan informasi yang tidak dapat diolah oleh komputer, permasalahan lainnya adalah *web* yang ada saat ini kebanyakan memiliki fitur yang sama. Namun, dalam pembuatannya setiap *web* hanya didesain khusus untuk pembuatan satu *web* saja. Tentu saja hal ini dapat dibantu dengan menerapkan paradigma *software product line* dimana *software product line* dapat menghemat biaya produksi, mempercepat waktu produksi ke pasar serta lebih menjadi kualitas dari produk yang dihasilkan (Pohl et al., 2005).

Pada tahun 2001, Berners-Lee et al. (2001) menemukan sebuah teknologi yang dinamakan *web* semantik, dimana *web* semantik akan membuat sebuah struktur untuk informasi yang terdapat pada *web* sehingga informasi-informasi tersebut dapat diolah oleh program komputer (Berners-Lee et al., 2001). Menurut Rob McCool, Format yang kompleks dan pengguna harus mengorbankan kemudahan ekspresivitas serta membayar biaya yang besar untuk translasi dan perawatan menjadi alasan kenapa *web semantic* tidak akan pernah diadopsi publik secara luas (Schoop et al., 2006).

Untuk mempermudah proses pembuatan *web* semantik, maka diciptakan *web* pragmatis dimana tujuannya adalah meningkatkan kolaborasi manusia untuk lebih efektif dengan teknologi yang tepat seperti sistem untuk negosiasi ontologi, interaksi bisnis yang terdapat pada ontologi, dan untuk membangun ontologi pragmatis

pada praktik masyarakat (Schoop et al., 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 pemrograman 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, 2016). Namun, pada penelitian tersebut pembentukan *business logic* dari *web* yang dibentuk masih secara manual. Hal ini dikarenakan pada ontologi tidak terdapat *business logic* sehingga tidak dapat secara otomatis. *Business logic* tersebut dapat diperoleh dari ABS *microservices* seperti pada Gambar 1.1.



Gambar 1.1: Roadmap pembentukan *web* berbasis semantik

Dengan memanfaatkan ABS *microservices* yang dihasilkan dari SPLE, maka tidak hanya membuat sebuah *web* berbasis semantik saja tetapi Zotonic yang dihasilkan nantinya dapat membuat beberapa *web* berbasis semantik yang memiliki

struktur yang sama yang berasal dari ontologi dan *business logic* yang dapat disesuaikan dengan kebutuhannya. Oleh karena itu, penting untuk dilakukan penelitian mengenai bagaimana cara menghubungkan ontologi tersebut dengan *web services* pada Zotonic agar lebih memudahkan dan lebih efisien dalam pembuatan *web*.

1.2 Perumusan Masalah

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

1. Bagaimana proses integrasi ontologi kepada *web service*?
2. Apakah dapat dikembangkan sebuah program *adaptor* yang akan melakukan integrasi ontologi dan *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 penelitian 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, perumusan masalah yang akan diteliti oleh penulis, tujuan penelitian, ruang lingkup penelitian serta sistematika penulisan.

- Bab 2 TINJAUAN PUSTAKA

Bab 2 berisi mengenai penjelasan terkait ontologi, *web framewok* berbasis semantic: Zotonic, dan *software product line*.

- Bab 3 RANCANGAN

Bab 3 berisi penjelasan mengenai rancangan dari sistem yang akan diimplementasikan, dimana bab ini terdiri atas 3 subbab, yaitu rancangan integrasi ontologi dan *web service*, rancangan *web service* serta rancangan *adaptor*.

- Bab 4 IMPLEMENTASI

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

- Bab 5 HASIL

Bab 5 berisi perubahan yang terjadi setelah eksperimen dan hasil uji coba 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

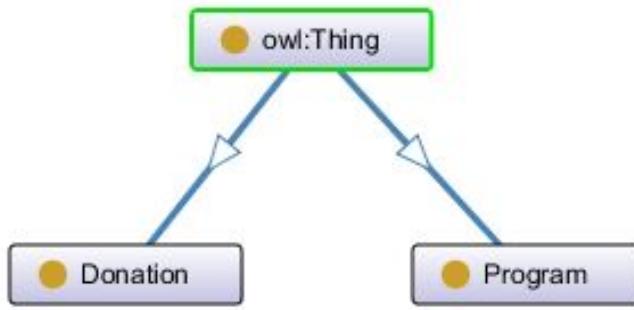
Ontologi merupakan bagian yang paling mendasar dari web semantik. Ontologi adalah sebuah deskripsi formal yang eksplisit tentang kelas (domain atau disebut juga sebagai *concepts*), relasi, dan *property* dari setiap domain, yang biasanya telah terdefinis secara baik (Hopkins dan Powell, 2015). Menurut 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.

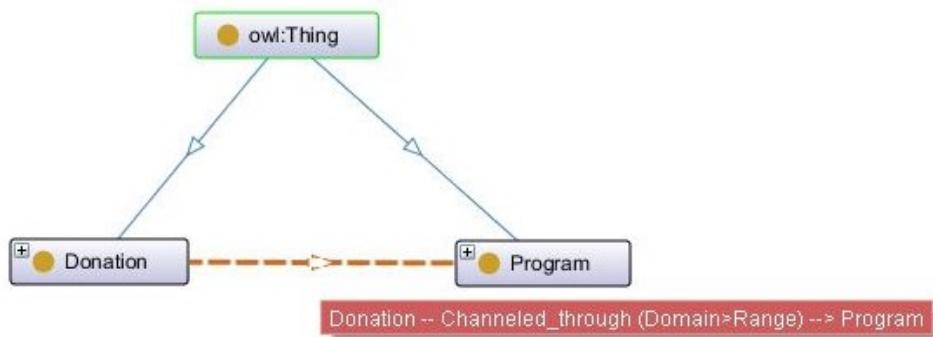
Ontologi terdiri dari beberapa komponen yaitu kelas, atribut, relasi. Ketika membuat ontologi, kelas merupakan komponen yang harus dibuat pertama kali karena kelas merupakan komponen utama dari ontologi. Kelas atau disebut juga domain, merupakan kumpulan data pada ontologi yang memiliki sebuah nama yang

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

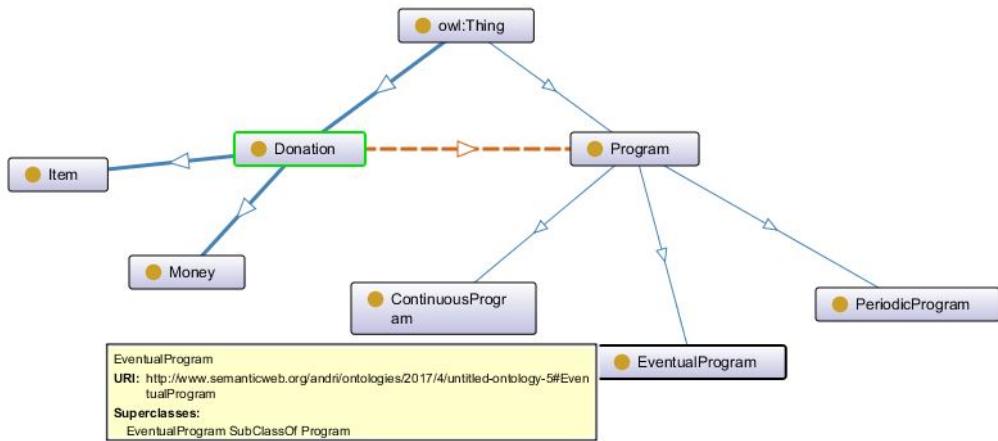


Gambar 2.2: Contoh Relasi pada Ontologi

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 hirarki yang dapat dimiliki oleh suatu kelas. Struktur hirarki 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 pada *Charity Organization*

Agar ontologi dapat dipahami oleh komputer, ontologi tersebut harus direpresentasikan dalam bentuk yang dapat dibaca oleh komputer salah satunya adalah OWL. *Ontology 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

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 perubahan kode berikutnya sehingga hal ini dapat membuat Zotonic lebih efisien dibandingkan PHP (Zotonic, ndc).

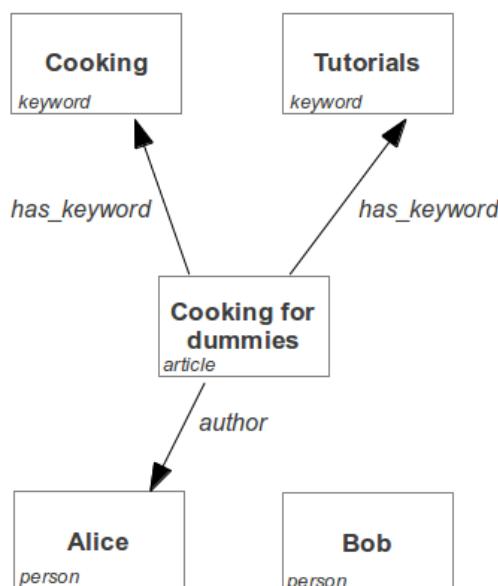
Zotonic juga menerapkan sistem modular. Zotonic dibuat dari lapisan kode yang umum dengan modul yang menyediakan fungsi utama lainnya. *Template*, *actions*, *controllers*, *javascript*, *css*, *dispatch rules*, semuanya berasal dari modul. Modul dapat memperbanyak, mengubah atau bekerja sama dengan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

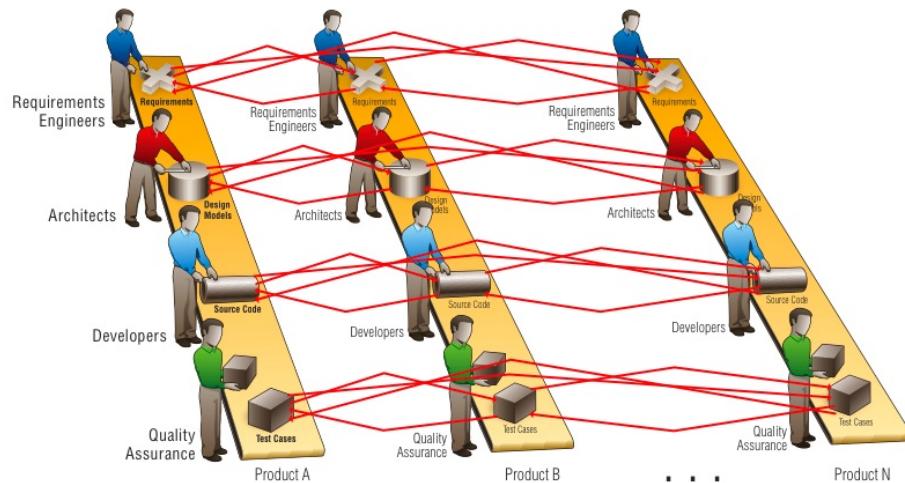
Pada Gambar 2.4, blok persegi menggambarkan *resource* dan panah menggambarkan *edge*. Alice adalah sebuah *resource* yang berkategori *person*. Begitu juga halnya dengan Bob yang merupakan *resource* dengan kategori *person*. Hal ini menunjukkan bahwa pada Zotonic, suatu kategori dapat memiliki banyak *instance* yang berupa *resource*. *Cooking* dan *Tutorials* sama-sama merupakan *resources* yang berkategori *keyword* sedangkan *Cooking For Dummies* merupakan *resource* yang berkategori *article*. *has_keyword* merupakan *edge* yang menunjukkan 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

Pada zaman dahulu, sebuah produk hanya akan dibuat untuk melayani satu pelanggan saja. Namun, seiring dengan perkembangan zaman dimana kemampuan membeli meningkat, maka semakin banyak orang yang mampu membeli suatu produk. Setelah beberapa saat kemudian, cara produksi terhadap suatu produk mulai berubah karena banyak orang yang berminat pada barang yang sama. Hal ini kemudian mendorong untuk dilakukan produksi massal pada produk yang akan diciptakan (Pohl et al., 2005). Memproduksi produk secara massal dengan standar yang diinginkan oleh pengguna tentu akan mengurangi biaya dan mempercepat waktu produksi jika harus membuat sebuah produk secara khusus. Istilah ini kemudian populer dengan sebutan *Production Line*.

Dengan populernya penggunaan *Production Line* pada industri, metode ini juga mulai diterapkan dalam industri teknologi dan dikenal dengan istilah *Software Product Line*. Hal ini karena dengan memanfaatkan *Software Product Line*, maka dapat mengurangi biaya untuk memproduksi produk, meningkatkan kualitas dari produk yang dihasilkan, serta mengurangi waktu dari produk tersebut untuk siap dipasarkan. Mengurangi biaya produk karena dengan menggunakan paradigma *software product line*, bagian dari produk sebelumnya dapat digunakan lagi untuk produk berikutnya (Pohl et al., 2005). Meningkatkan kualitas dari produk yang dihasilkan karena produk yang dihasilkan akan digunakan oleh banyak pengguna sehingga kesempatan untuk mendeteksi *bug* agar dapat diperbaiki lebih besar (Pohl et al., 2005). Dan terakhir, mengurangi waktu produk untuk siap dipasarkan

karena produk baru yang akan dibuat dapat menggunakan bagian atau bahkan semua bagian dari produk sebelumnya.



Gambar 2.5: Proses pembuatan produk menggunakan metode SPL

Sumber gambar: http://www.biglever.com/images/solution/Productionline_post.jpg

Seperti yang terlihat pada Gambar 2.5, pada proses produksi menggunakan SPL, hasil dari produk sebelumnya (pada gambar merupakan produk A) akan kembali digunakan untuk memproduksi produk selanjutnya. Bagian yang digunakan dapat berupa seluruh komponen dari produk pertama, ataupun hanya beberapa bagian yang digunakan kembali. Selanjutnya ketika produk berikutnya selesai (misal produk B), maka tambahan yang dibuat pada produk tersebut dapat digunakan kembali pada produk selanjutnya juga.

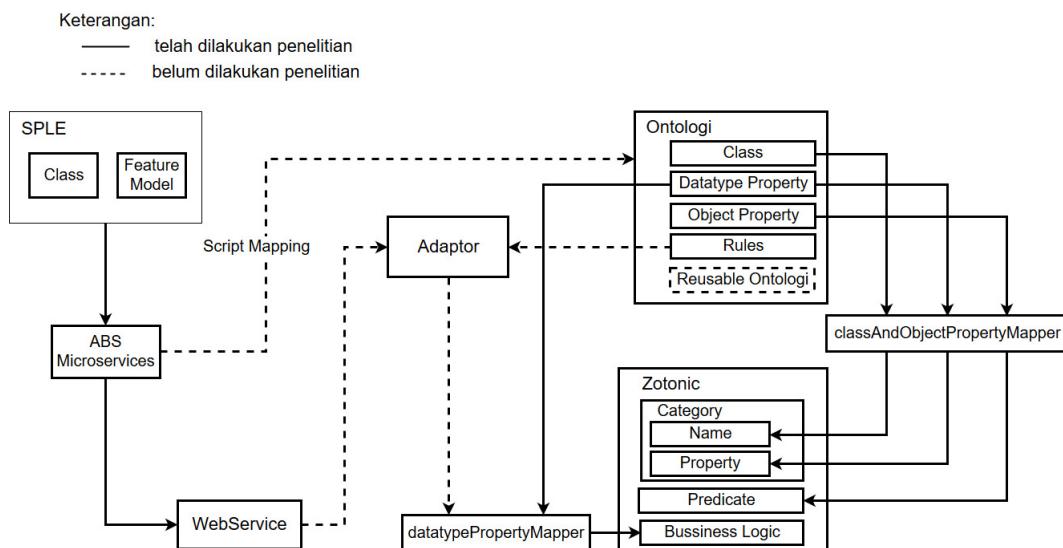
BAB 3

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*

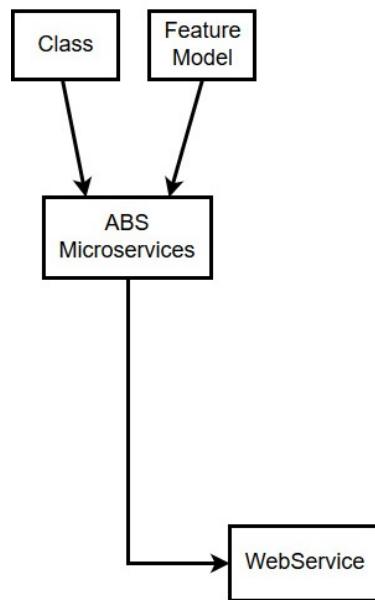
Supaya dapat menghasilkan program yang bertujuan untuk melakukan integrasi ontologi dan *web services* maka perlu dirancang secara keseluruhan bagaimana program ini akan bekerja. Sesuai dengan kebutuhan dari program, maka integrasi ini akan dilakukan pada sebuah *framework* sekaligus CMS Zotonic, yang 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

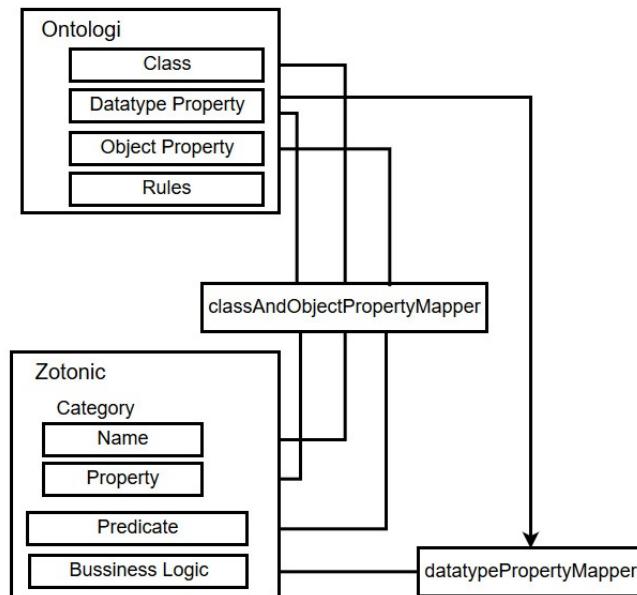


Gambar 3.2: Rancangan Bagian ABS

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 digambar pada Gambar 3.3 berikut ini.



Gambar 3.3: Rancangan Bagian Zotonic dan Ontologi

Namun pada penelitian tersebut, pembuatan *business logic* masih bersifat manual dimana kode *business logic* 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

Supaya *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 <i>error exception</i>

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

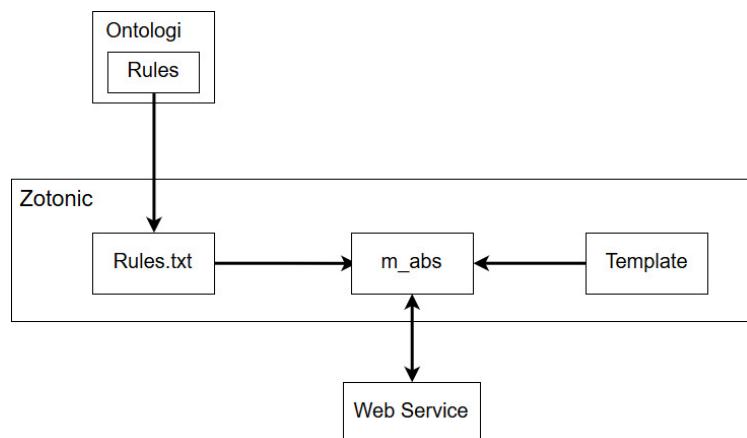
Status	Data
200	Data yang ingin dikirim
201	Data keseluruhan
400	null

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 berkas 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],
  "checkExistProgram" : ["http://54.169.128.6:8080/abs/program/
    checkExist", 1],
  "checkExistDonation" : ["http://54.169.128.6:8080/abs/donation/
    checkExist", 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 memanfaatkan *adaptor* yang sudah diimplementasikan pada *business logic*.

4.1 *Interface Adaptor*

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 4.1: Fungsi yang harus diekspor untuk *model*

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

Seperti yang dapat dilihat pada Kode 4.1, 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 4.2 berikut

Kode 4.2: 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 4.3 berikut

Kode 4.3: 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 4.4: 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 4.4, 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 4.5: 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 ->
            {DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode
                ({Data})),
            lager:info("[ABS] result ~p", [DecodeJson]),
            case proplists:get_value(<<"status">>, DecodeJson) of
                200 ->
                    proplists:get_value(<<"data">>, DecodeJson),
                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 4.5, 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 (<i>insert, update, delete</i>)	Message
400	<i>Error</i> dari API	Message
<i>Other</i>	<i>error</i>	Message

4.2 Implementasi *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 4.6: Struktur tabel `rules`

```
{
  nama_fungsi : [endpoint, jumlah_parameter]
}
```

Setelah tabel `rules` selesai dibuat sesuai dengan struktur pada Kode 4.6, tabel `rules` tersebut disimpan ke dalam sebuah file yang bernama `rules.txt` dan ditaruh pada `root` dari Zotonic. Selain `rules.txt`, pada *adaptor* juga perlu diimplementasikan fungsi-fungsi berikut agar *adaptor* dapat bekerja.

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.
```

Pada Kode 4.10, terdapat *specifications* untuk fungsi fetch_data dimana fungsi ini menerima dua parameter yaitu *url* atau *endpoint web services* yang ingin dituju serta *Query* yang berisikan parameter dalam bentuk *json* yang akan dikirim. Fungsi fetch_data ini kemudian akan mengembalikan sebuah *list* hasil dari pemanggilan *web services* tersebut. Fungsi fetch_data akan memanggil fungsi post_page_body yang akan menjalankan pemanggilan ke *web service*. Jika fungsi

`post_page_body` mengembalikan *Json*, maka *Json* tersebut akan *didecode* menggunakan `jiffy:decode/1` sehingga dihasilkan sebuah *list* yang akan dikirim kepada pemanggilan fungsi `fetch_data`. Tetapi jika fungsi `post_page_body` mengembalikan *error*, maka fungsi `fetch_data` juga akan meneruskan *error* tersebut kepada fungsi pemanggilnya. Adapun implementasi dari fungsi `post_page_body` sebagai Kode 4.11 berikut.

Kode 4.11: Implementasi fungsi `post_page_body`

```
post_page_body(Url, Body) ->
  case httpc:request(post, {Url, [], "application/json", Body},
  [], []) of
    {ok, {_, _, Response}} ->
      Response;
    Error ->
      {error, Error}
  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 dibutuhkan 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 4.12: Fungsi total sebelum *refactoring*

```

{%
  javascript %
  var predId = [];
  var total = 0;
{%
  endjavascript %
{%
  with m.searchpaged[{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 4.12, 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 4.12 akan diganti menjadi Kode 4.13

Kode 4.13: Fungsi total setelah refactoring

```

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

```

Pada Kode 4.13, 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 berkas baru yang ditambahkan ke dalam struktur Zotonic. Berikut adalah perbandingan struktur dari Zotonic yang telah dapat membuat struktur Zotonic dari ontologi sebelum penambahan berkas baru untuk menjalankan fungsi integrasi ontologi dan *web service* dan struktur dari *default* Zotonic pada Tabel 5.1

Tabel 5.1: Perbandingan struktur Zotonic yang asli dan hasil penelitian sebelumnya

Zotonic Default	Hasil penelitian sebelumnya
.rebar/	.rebar/
bin/	bin/
deps/	deps/
doc/	doc/
docker/	docker/
ebin/	ebin/
include/	include/
modules/	modules/
priv/	priv/
src/	src/
user/	user/
.dockerignore	.dockerignore
.editorconfig	.editorconfig
.travis.yml	.travis.yml
AUTHORS	AUTHORS
CONTRIBUTING.md	CONTRIBUTING.md
CONTRIBUTORS	CONTRIBUTORS
Dockerfile	Dockerfile
Dockerfile.dev	Dockerfile.dev
Dockerfile.heavy	Dockerfile.heavy
GNUmakefile	GNUmakefile
LICENSE	LICENSE

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

Setelah implementasi, perlu ditambahkan berkas rules.txt pada folder Zotonic dimana berkas 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 berkas /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: Konfigurasi berkas /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 berkas 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

```

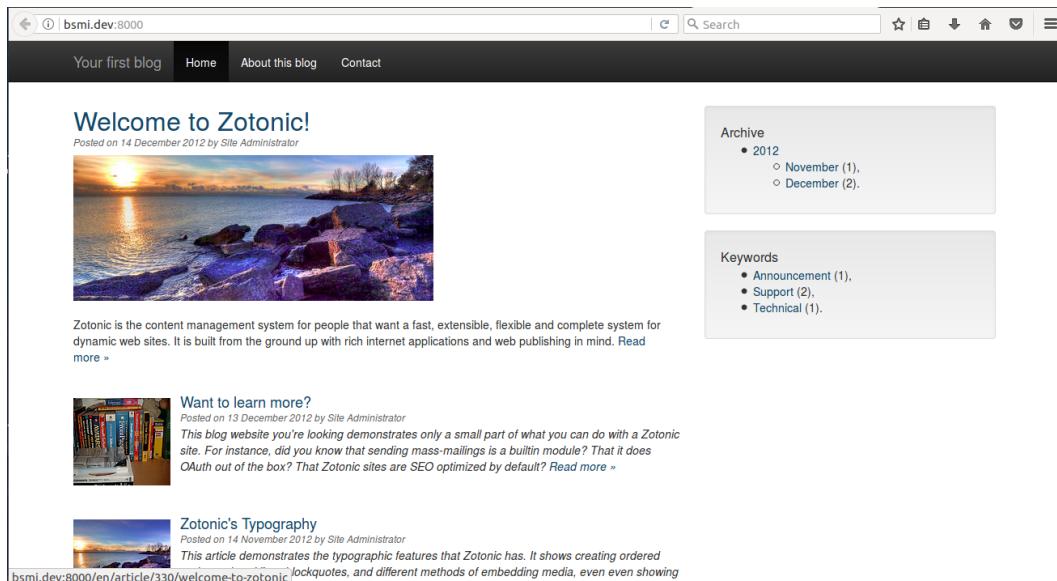
...
<p class="programtotal" id="programtotal">
    <b>{_ total _}</b> :
</p>
{%
    javascript %
        var total = {{m.abs.totalDonation[query id=id]}};;
        if (total == 0) {
            document.getElementById("programtotal").className += "hidden";
        }
        else {
            document.getElementById("programtotal").innerHTML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

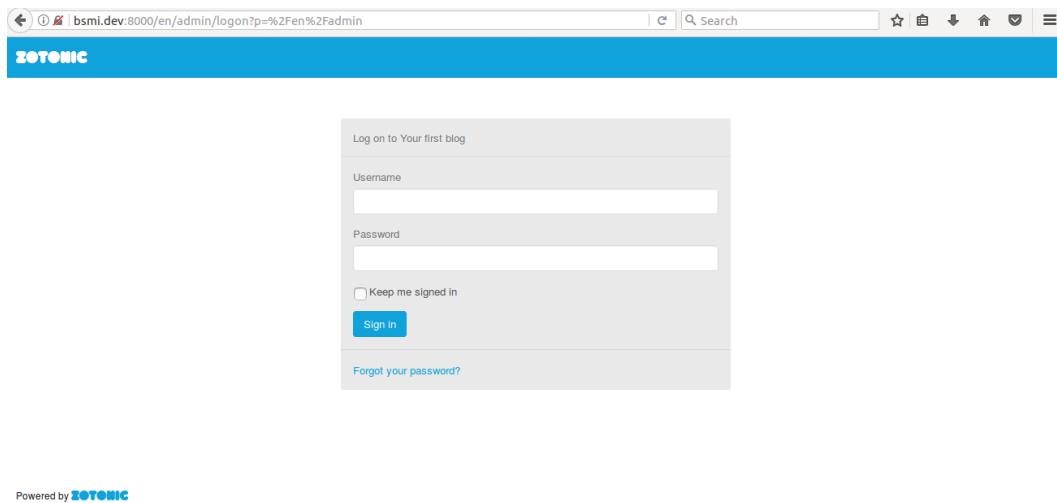
5.3 Contoh Penerapan pada Web BSMI

Setelah membuat *site* baru pada Zotonic, tampilan awal dari *site* tersebut seperti pada gambar Gambar 5.1. Tampilan *homepage* dari *site* tersebut telah menggunakan *bootstrap* sebagai css.



Gambar 5.1: Tampilan awal *site* pada Zotonic

Selain *homepage*, Zotonic sebagai sebuah CMS juga telah menyediakan halaman admin untuk pengguna agar dapat mengatur isi dari *site* tersebut. Untuk dapat masuk ke dalam halaman admin, pengguna dapat mengakses halaman admin pada `namasite.dev:8000/en/admin`. Setelah mengakses halaman tersebut, pengguna akan melihat halaman untuk *logon* seperti pada Gambar 5.2. Akun *default* untuk *logon* sebagai admin adalah *username* admin dengan *password* admin.



Gambar 5.2: Tampilan login admin pada Zotonic

Setelah berhasil untuk *logon*, pengguna akan melihat halaman *dashboard* admin seperti pada Gambar 5.3. Pada *dashboard* admin ini, sama seperti CMS lainnya dimana pengguna dapat melakukan berbagai hal seperti menambahkan konten dari *site*, menambahkan *user* baru untuk *logon*, mengatur *module-module* yang digunakan pada *site*, dan lain-lain.

The screenshot shows the Zotonic admin dashboard with a blue header bar. The header includes the Zotonic logo, navigation links for Dashboard, Content, Structure, Modules, Auth, System, and a search bar. Below the header is a toolbar with buttons for 'Make a new page or media', 'All pages', 'All media', and 'All page connections'. The main content area is titled 'Dashboard' and contains several sections: 'Latest modified texts' (listing 'Zotonic's Typography', 'Want to learn more?', 'Welcome to Zotonic!', 'Contact', and 'About this blog' with 'Article' and 'Text' types), 'Latest modified locations' (listing 'No items'), 'Latest modified events' (listing 'No items'), and 'Latest modified media items' (listing 'Zotonic introduction video' (Media), 'A bunch of computer books' (Image), and 'Rocky sunrise' (Image)). Each item has 'view' and 'edit' buttons.

Gambar 5.3: Tampilan *dashboard* admin pada Zotonic

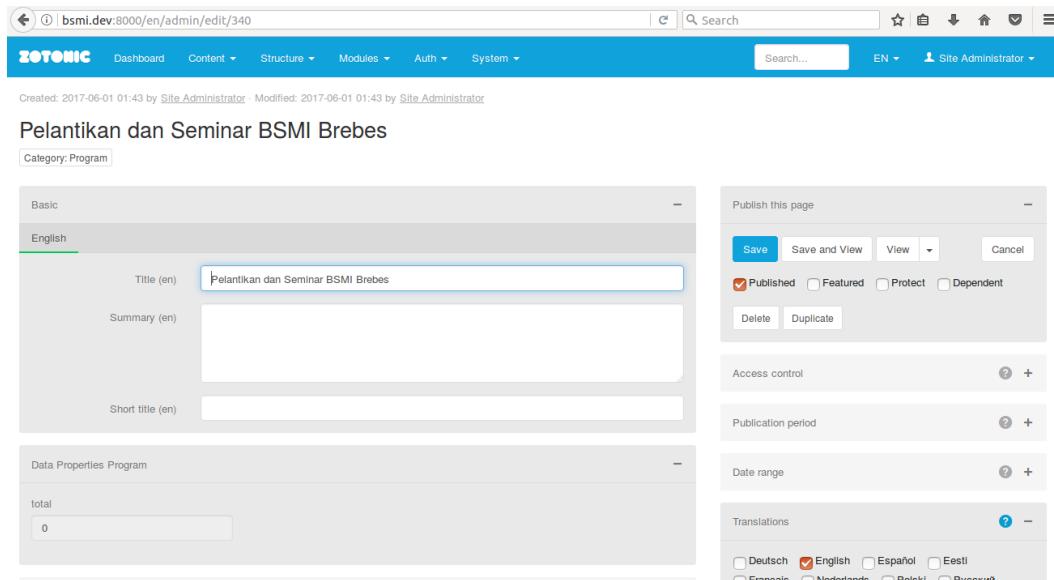
Untuk dapat melihat apakah hasil implementasi pada bab sebelumnya sudah bekerja atau belum, maka perlu dibuat sebuah *page* baru dengan kategori program pada *site*. Hal ini dapat dilakukan dengan menekan tombol berwarna biru dengan tulisan *Make a new page or media* pada *dashboard* admin. Selanjutnya akan muncul *modals* untuk membuat *page* baru seperti pada gambar Gambar 5.4. Setelah mengisi nama dan memilih kategori program, *checklist* pada *published* agar *page* dapat ditampilkan pada site. Selanjutnya klik tombol *Make page*.

The screenshot shows the 'Make a new page' modal window over the Zotonic admin dashboard. The modal has tabs for 'Create Page', 'Upload File', 'Upload by URL', and 'Embed URL'. The 'Create Page' tab is selected. It contains fields for 'Page title' (set to 'Pelantikan dan Seminar BSMI Brebes'), 'Category' (set to 'Program'), and 'Published' (with a checked checkbox). At the bottom of the modal are 'Cancel' and 'Make page' buttons. The background dashboard shows the same sections as in Gambar 5.3, including 'Latest modified texts', 'Latest modified locations', 'Latest modified events', and 'Latest modified media items'.

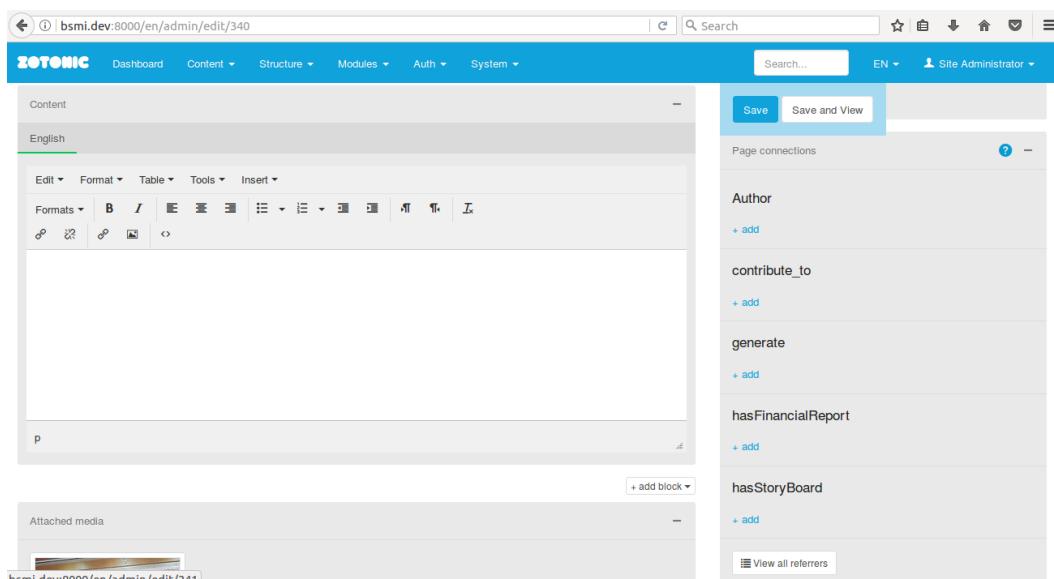
Gambar 5.4: Tampilan membuat *page* pada Zotonic

Selanjutnya pengguna akan dibawa menuju sebuah halaman untuk mengubah detail dari *page* yang dibuat seperti pada Gambar 5.5. Program memiliki sebuah

data properties yaitu total, dimana total akan diubah otomatis menggunakan *business logic* berdasarkan donasi yang ada. Namun karena belum terdapat donasi yang terhubung pada program ini, sehingga nilai dari total bernilai 0. Pada halaman ini pengguna dapat menambahkan informasi seperti *summary* dari *page* yang dibuat atau menambahkan konten dari *page* dan gambar seperti Gambar 5.6. Selain itu, pada halaman ini pengguna juga dapat menambahkan *page connection* agar halaman yang sedang dibuat dapat memiliki keterkaitan dengan halaman lainnya.



Gambar 5.5: Tampilan halaman mengubah detail *page* kategori program



Gambar 5.6: Tampilan halaman mengubah detail *page* kategori program

Setelah selesai mengubah detail dari *page*, pengguna dapat menyimpan peruba-

han yang telah dibuat dengan menekan tombol *save*. Selain tersimpan pada database Zotonic, *page* akan disimpan juga pada sebuah database yang didesain sesuai dengan ontologi yang ada. Penyimpanan *page* ke database tersebut dengan memanfaatkan *adaptor* yang telah dibuat. Pada Gambar 5.7, dapat dilihat bahwa *page* dengan judul Pelantikan dan Seminar BSMI Brebes yang memiliki id 340 yang dibuat telah tersimpan pada database tersebut. Hasil dari *page* yang dibuat dapat dilihat pada Gambar 5.8.

The screenshot shows the phpMyAdmin interface connected to a MySQL database named 'abs_program'. The 'program' table is selected. The table has two columns: 'id' and 'name'. There is one row with id 340 and name 'Pelantikan dan Seminar BSMI Brebes'. The interface includes various navigation tabs like 'Browse', 'Structure', 'SQL', and 'Operations'.

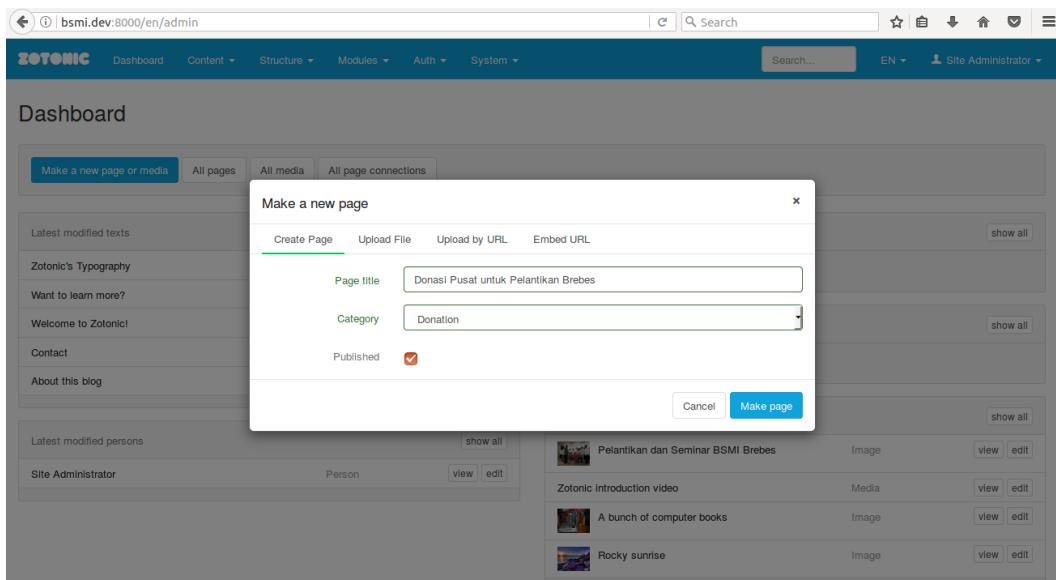
Gambar 5.7: Penyimpanan *page* program pada database eksternal

The screenshot shows a web browser displaying a blog post titled 'Pelantikan dan Seminar BSMI Brebes'. The post features a large image of a group of people at an event. To the right of the post is a sidebar with an 'Archive' section showing entries for 2012 (November 1, December 2) and a 'Keywords' section listing 'Announcement (1)', 'Support (2)', and 'Technical (1)'. At the bottom of the page is a note: '*Brebes adalah salah satu daerah rawan bencana. Kami mengucapkan selamat dan mengapresiasi kepada BSMI'.

Gambar 5.8: Tampilan *page* Pelantikan dan Seminar BSMI Brebes

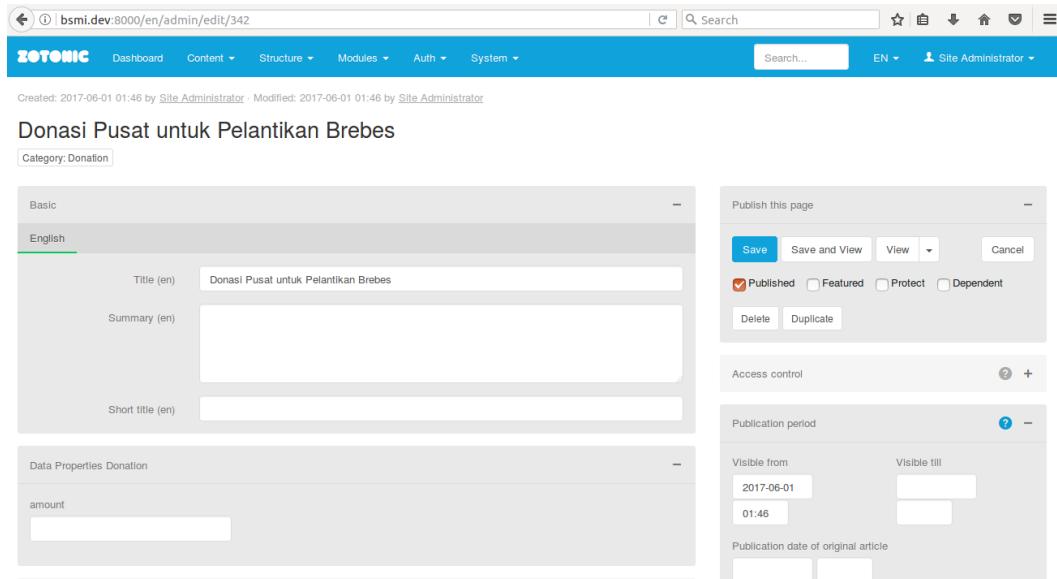
Selanjutnya untuk mencoba *business logic* yang telah diimplementasikan, perlu terlebih dahulu untuk membuat *page* dengan kategori donasi. Hal ini karena setiap

donasi akan terhubung dengan sebuah program sesuai dengan ontologinya. Proses pembuatan *page* untuk kategori donasi hampir sama dengan proses pembuatan *page* kategori lainnya yaitu dengan menekan tombol berwarna biru pada halaman *dashboard* admin yang bertulisan *Make a new page or media*, lalu mengisi judul dari *page* yang akan dibuat dan memilih kategori *donation* sebagai kategori dari *page* tersebut lalu klik tombol *Make page* seperti pada Gambar 5.9

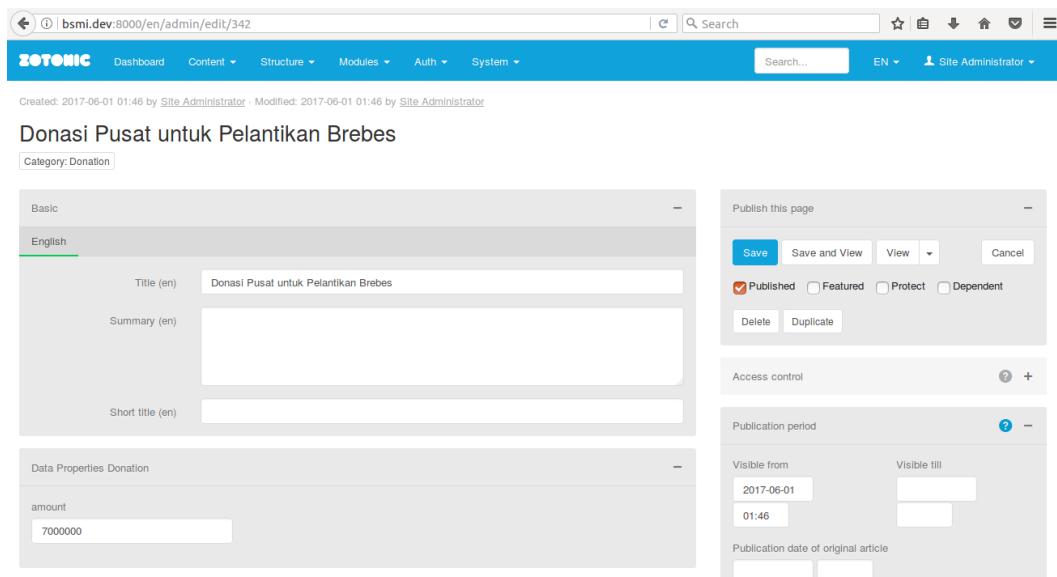


Gambar 5.9: Tampilan membuat *page* pada Zotonic

Selanjutnya pengguna akan menuju sebuah halaman untuk mengubah detail dari *page* yang sedang dibuat seperti pada Gambar 5.10. Donasi memiliki sebuah *data properties* yaitu *amount* dimana nilai *amount* ini nantinya akan diakumulasikan menjadi nilai dari total pada program yang terhubungan dengan *page* donasi ini. Untuk dapat melihat apakah *business logic* sudah bekerja, maka *amount* dari *page* donasi harus diisi dengan angka. Seperti pada Gambar 5.11, nilai dari *amount* diisi dengan 7000000. Nilai 7000000 ini akan menjadi bagian dari nilai total program yang terkait dengan donasi ini.

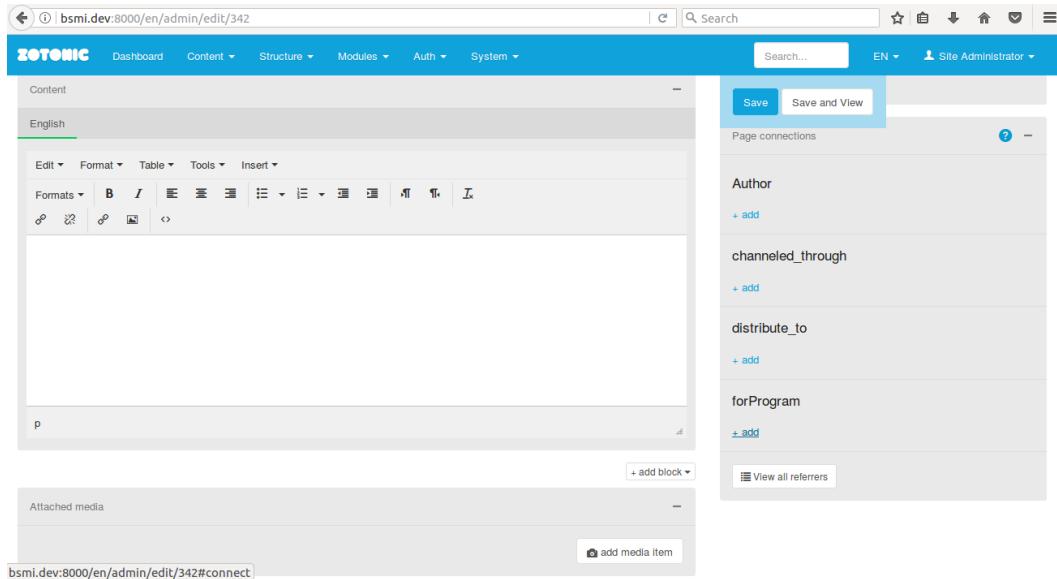


Gambar 5.10: Tampilan halaman untuk mengubah donasi pada Zotonic



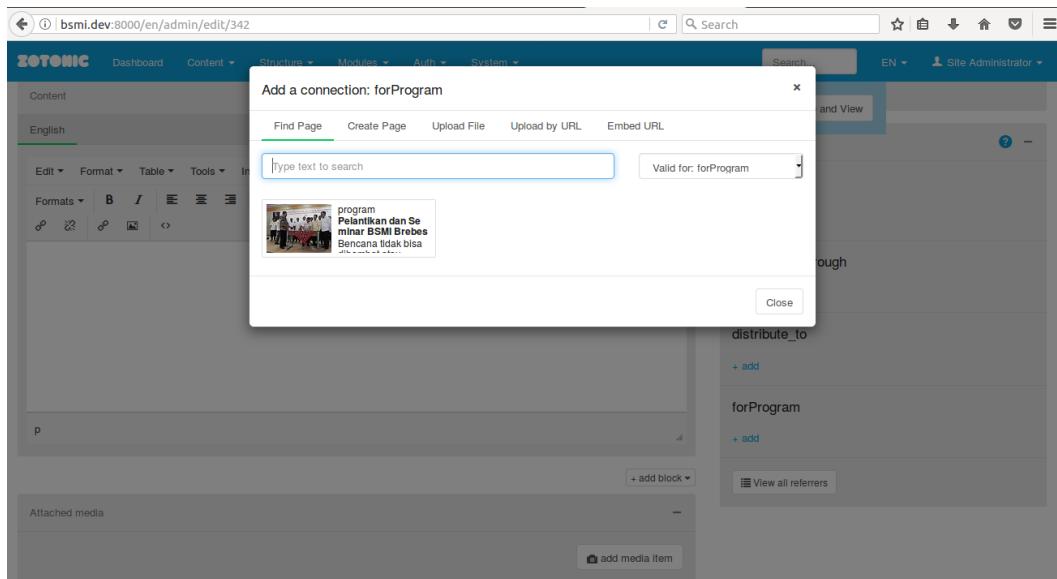
Gambar 5.11: Tampilan halaman mengubah detail *page* kategori donasi

Untuk membuat koneksi antara suatu donasi dengan suatu program, maka pada bagian *page connection* harus diisi koneksi *forProgram* yang menggambarkan donasi ini ditujukan untuk program yang mana seperti pada Gambar 5.12. Tekan *add* pada bagian *forProgram* untuk memilih program yang dituju, lalu akan muncul *modals* seperti pada Gambar 5.13.



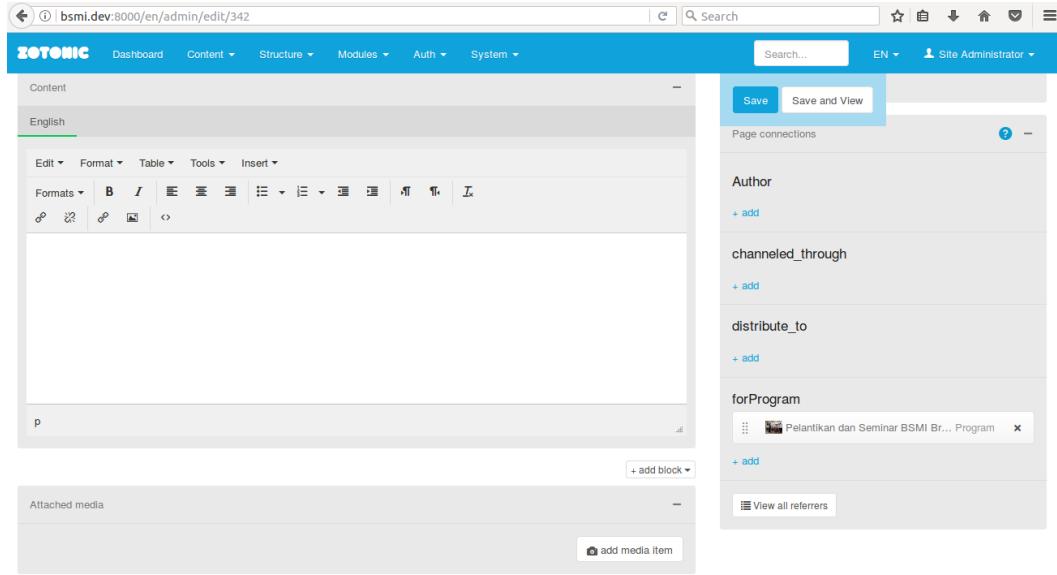
Gambar 5.12: Tampilan halaman mengubah detail *page* kategori donasi

Pada Gambar 5.13, *modals* yang muncul akan menampilkan seluruh *page* program yang telah ada. Pada contoh ini, karena *site* baru memiliki sebuah *page* dengan kategori program maka yang ditampilkan hanya satu. Dengan bantuan *modals* ini, pengguna akan lebih mudah untuk menambahkan koneksi antara donasi dan program.



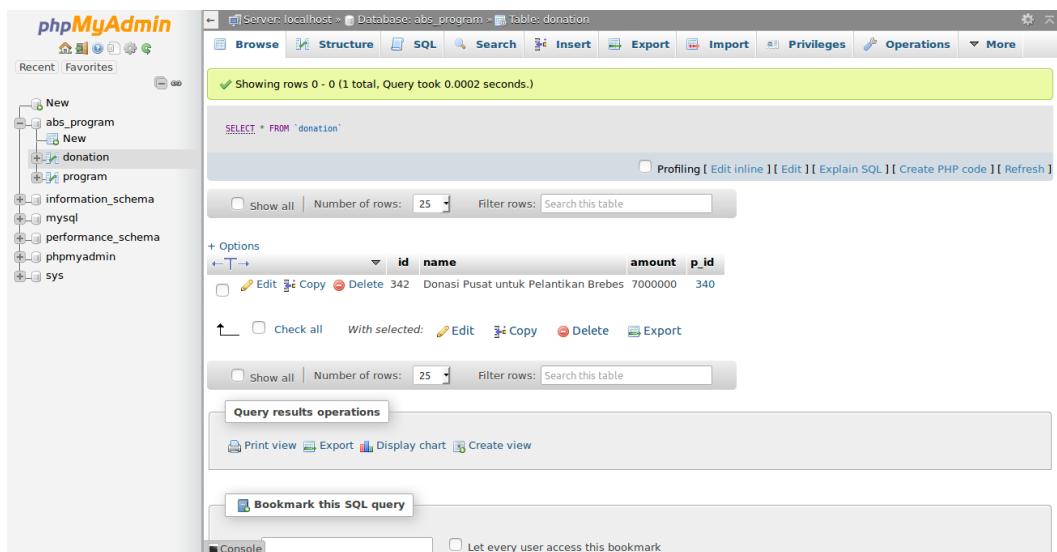
Gambar 5.13: Tampilan halaman mengubah detail *page* kategori donasi

Setelah memilih program mana yang akan ditujukan untuk donasi ini, tampilannya akan menjadi seperti pada Gambar 5.14. Lalu simpan perubahan dengan menekan tombol save.



Gambar 5.14: Tampilan halaman mengubah detail *page* kategori donasi

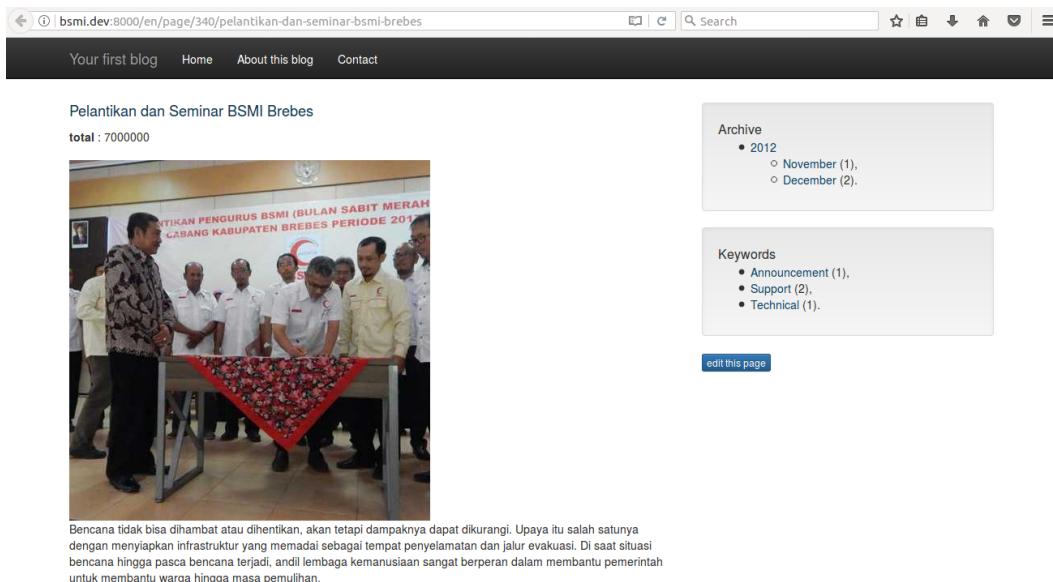
Ketika donasi tersebut disimpan, donasi tersebut akan disimpan juga pada database eksternal. Sama seperti dengan *page* kategori program, *page* kategori donasi juga akan disimpan ke database eksternal dengan memanfaatkan *adaptor* yang sudah diimplementasikan. Pada Gambar 5.15 dapat dilihat bahwa donasi dengan nama Donasi pusat untuk Pelantikan Brebes dengan amount sebesar 7000000 dan id 342 telah tersimpan. Selain itu, informasi tentang program yang terkait dengan donasi ini juga disimpan yang dilambangkan oleh p_id. Untuk donasi ini, karena donasi dengan id 342 ditujukan untuk program Pelantikan dan Seminar BSMI Brebes yang memiliki id 340 maka p_id dari donasi ini adalah 340.



Gambar 5.15: Penyimpanan donasi pada database eksternal

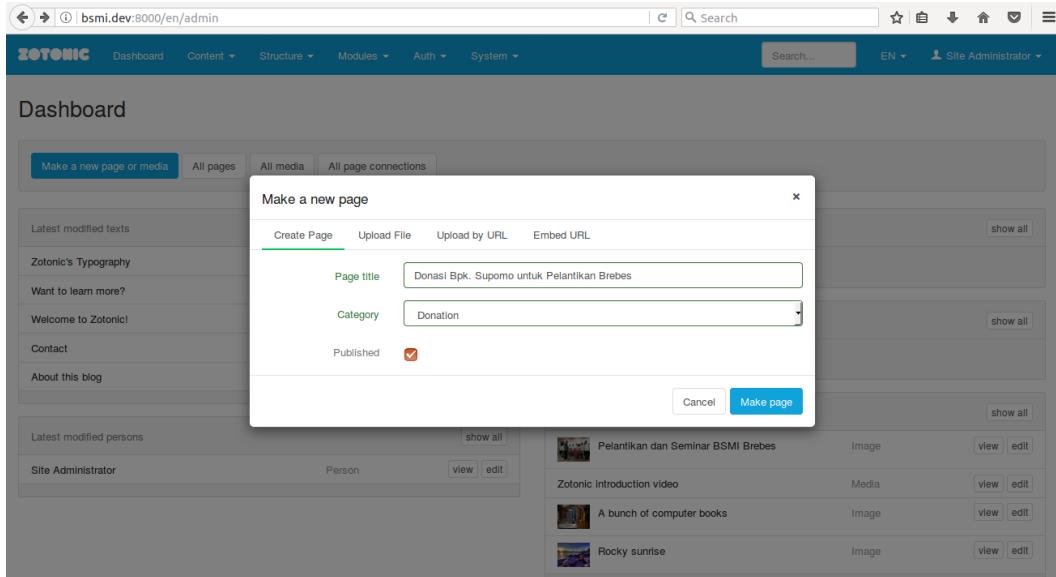
Setelah pembuatan donasi yang ditujukan untuk program Pelantikan dan Sem-

inar BSMI Brebes, maka terjadi perubahan pada halaman program Pelantikan dan Seminar BSMI Brebes yaitu tampilnya nilai dari total donasi untuk program tersebut yang dapat dilihat pada Gambar 5.16. Perhitungan jumlah donasi tersebut telah memanfaatkan *adaptor* yang telah diimplementasikan.



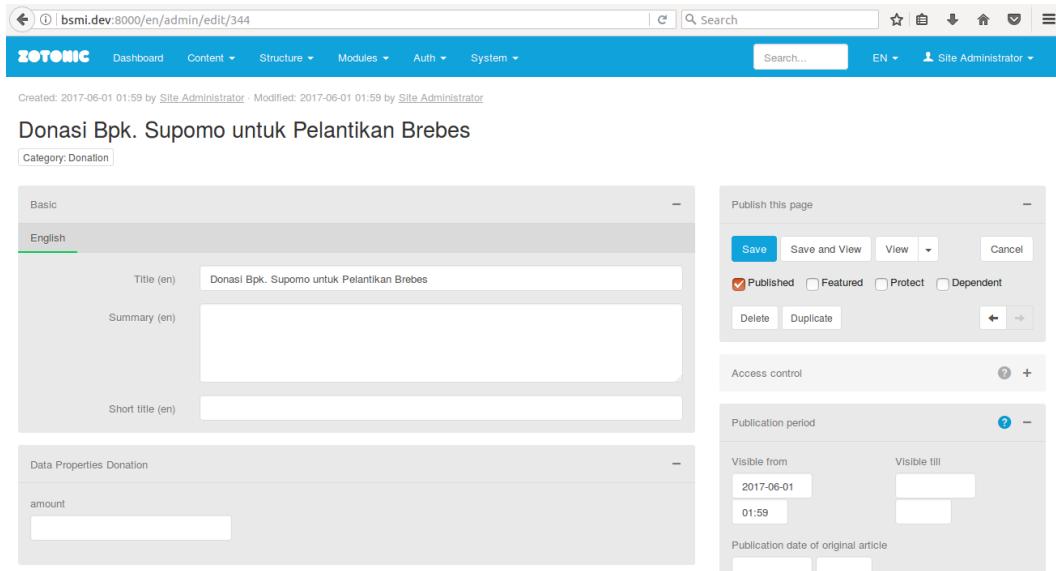
Gambar 5.16: Tampilan *page* Pelantikan dan Seminar BSMI Brebes setelah pembuatan donasi

Agar lebih dapat terlihat apakah *business logic* untuk total donasi telah benar, maka ditambahkan lagi donasi baru untuk program Pelantikan dan Seminar BSMI Brebes. Sama seperti pembuatan donasi sebelumnya, langkah pembuatan donasi baru dengan menekan tombol *Make a new page or media* pada halaman *dashboard* admin. Lalu akan muncul *modals*. Pada *modals* tersebut perlu diisi nama dari donasi yang akan dibuat dan kategorinya dipilih *donation* seperti Gambar 5.17. Setelah selesai mengisi informasi tersebut, dapat menekan tombol *Make page* pada *modals* tersebut.



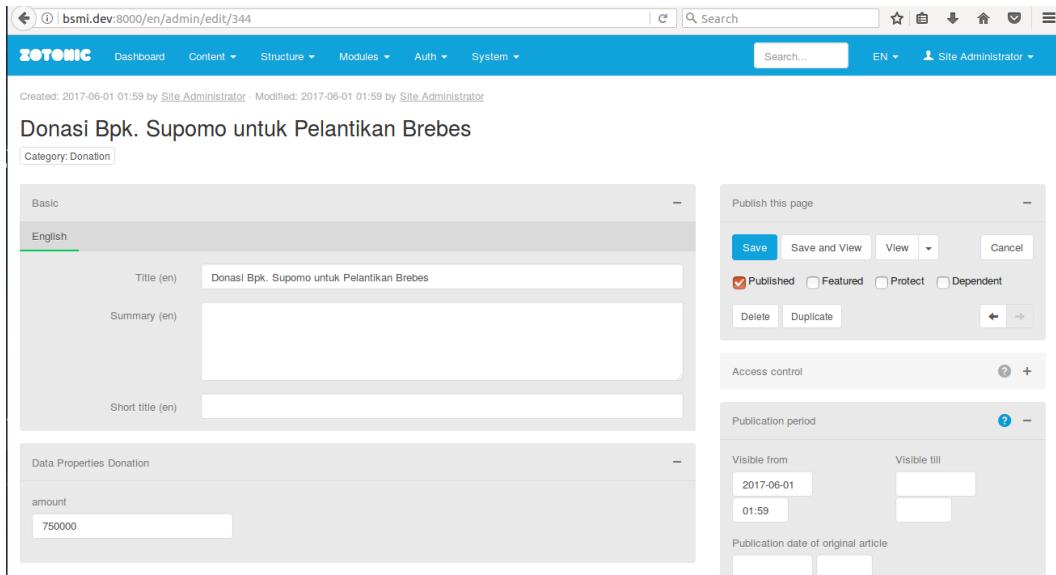
Gambar 5.17: Tampilan membuat *page* pada Zotonic

Setelah menekan tombol *Make page* pada *modals* tersebut, pengguna akan diarahkan menuju halaman untuk mengubah detail dari *page* yang sedang dibuat seperti pada Gambar 5.18. Sama seperti pada proses pembuatan *page* donasi sebelumnya, pengguna harus memasukkan nilai *amount* dari donasi pada *data properties* seperti Gambar 5.19.



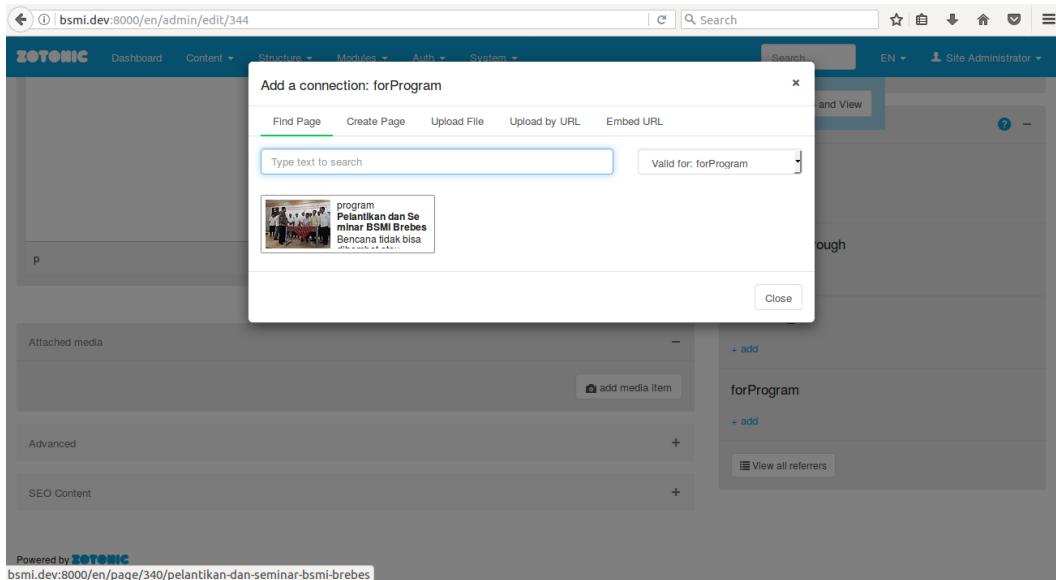
Gambar 5.18: Tampilan halaman mengubah detail *page* pada kategori Donasi

Untuk donasi dengan judul Donasi Bpk. Supomo untuk Pelantikan Brebes, nilai *amount* diberi 750000 seperti Gambar 5.19. Setelah mengisi nilai *amount*, selanjutnya pengguna harus menghubungkan donasi dengan suatu program. Untuk melakukan hal tersebut, tekan tombol *add forProgram* pada *page connection*.

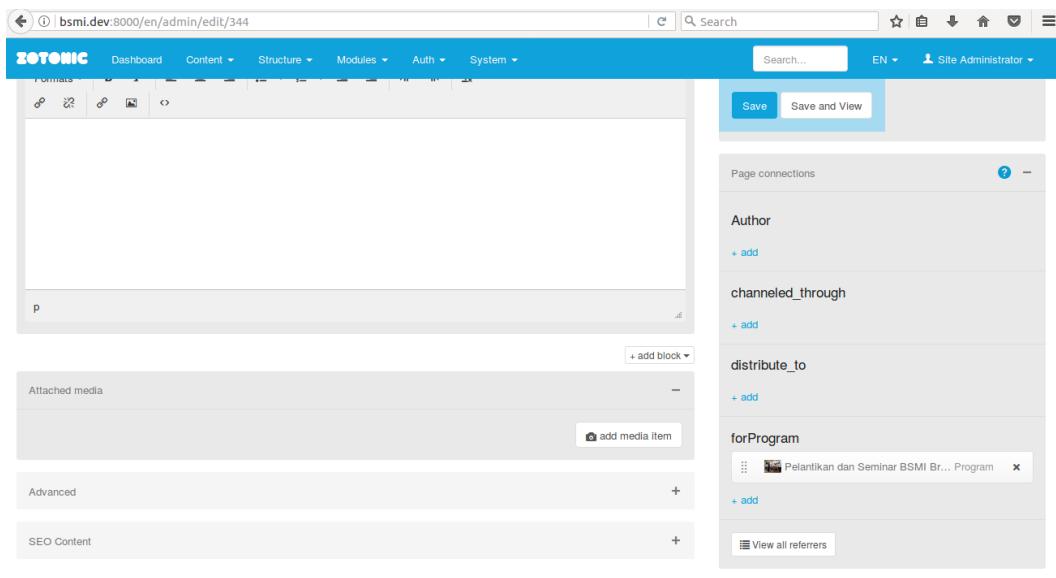


Gambar 5.19: Tampilan halaman mengubah detail *page* pada kategori Donasi

Lalu akan muncul sebuah *modals* yang menampilkan seluruh program yang ada seperti pada Gambar 5.20. Pilih program mana yang akan dituju, lalu tekan tombol *close* sehingga akan muncul tampilan seperti Gambar 5.21. Setelah proses perubahan selesai dilakukan, tekan tombol *save* untuk menyimpan perubahan yang telah dibuat.

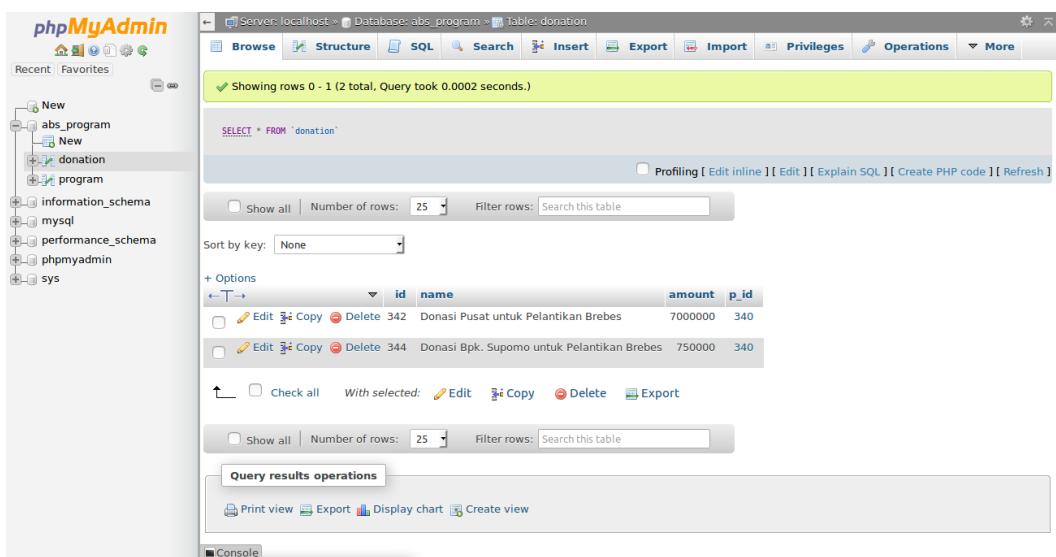


Gambar 5.20: Tampilan halaman mengubah detail *page* pada kategori Donasi



Gambar 5.21: Tampilan halaman mengubah detail *page* pada kategori Donasi

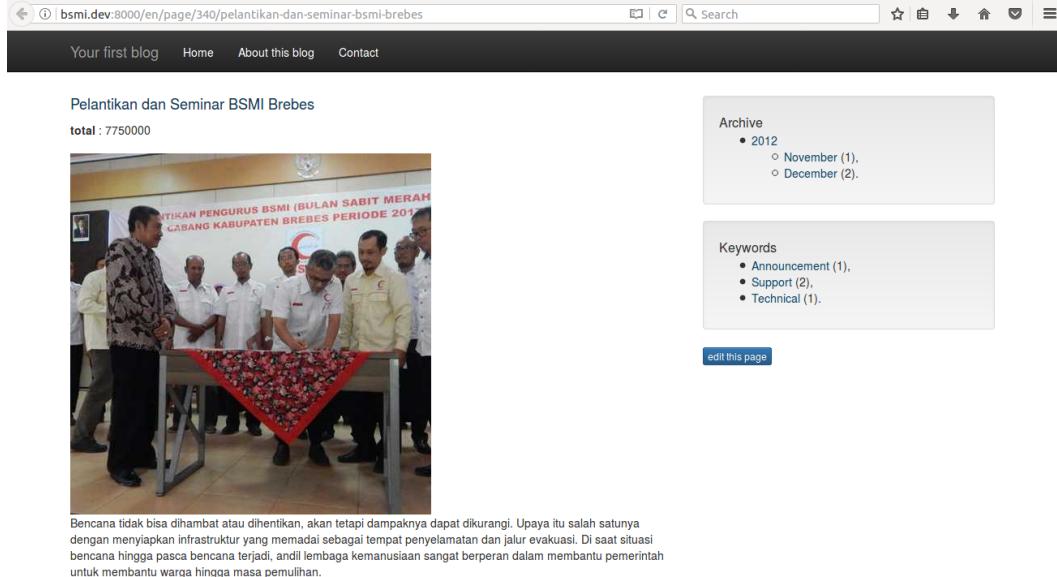
Sama seperti donasi sebelumnya, ketika sebuah donasi disimpan maka akan tersimpan juga pada database eksternal dengan memanfaatkan *adaptor* yang telah diimplementasi. Hal ini dapat dilihat pada Gambar 5.21 dimana donasi dengan judul Donasi Bpk. Supomo untuk Pelantikan Brebes yang memiliki id 344 dan *amount* 750000 telah tersimpan pada database eksternal. Karena donasi ini ditujukan untuk program Pelantikan dan Seminar BSMI Brebes yang memiliki id 340, maka p_id dari donasi ini adalah 340.



Gambar 5.22: Penyimpanan donasi pada database eksternal

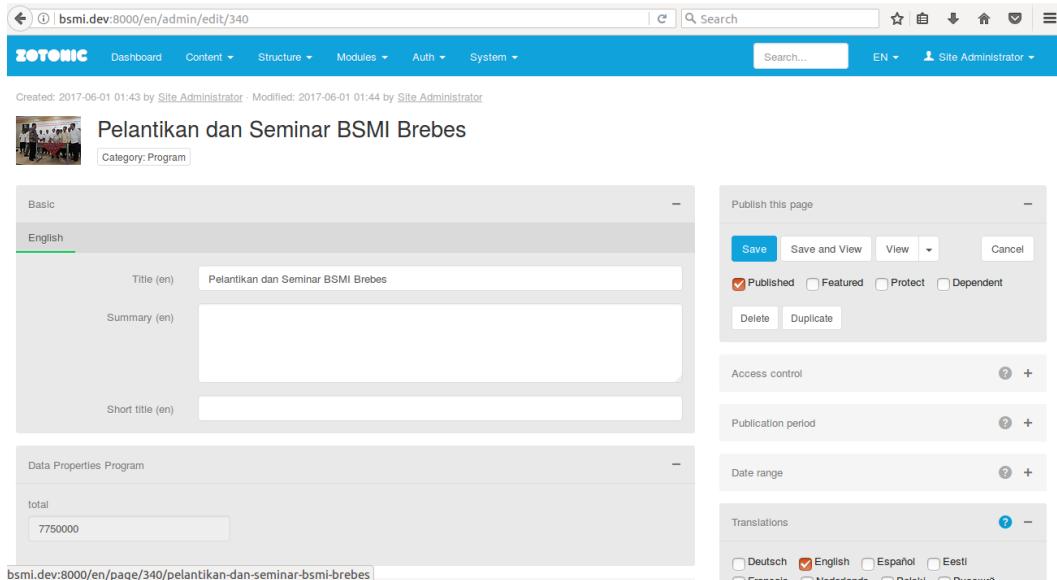
Setelah donasi kedua berhasil disimpan, maka total donasi untuk program "Pelantikan dan Seminar BSMI Brebes" akan berubah menjadi 7750000 dimana donasi "Donasi Pusat untuk Pelantikan Brebes" sebesar 7000000 dan donasi

"Donasi Bpk. Sutomo untuk Pelantikan Brebes" sebesar 750000. Hal ini dapat dilihat pada *page* Pelantikan dan Seminar BSMI Brebes seperti pada Gambar 5.23 dimana totalnya sudah berubah menjadi 7750000.



Gambar 5.23: Tampilan *page* Pelantikan dan Seminar BSMI Brebes setelah penambahan donasi

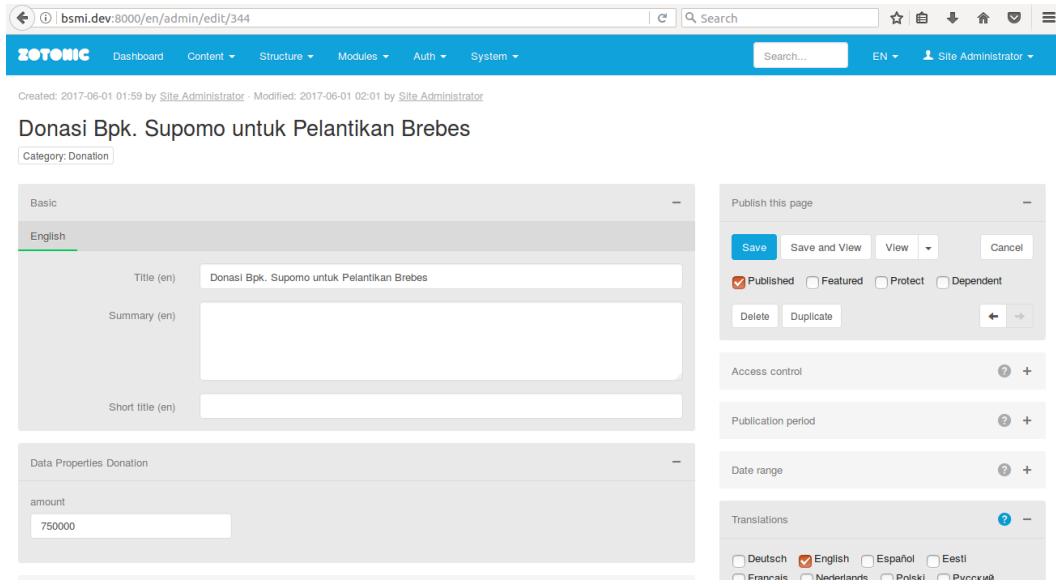
Selain pada *page* Pelantikan dan Seminar BSMI Brebes, perubahan juga terjadi pada halaman untuk mengubah detail *page* untuk program tersebut. Seperti pada Gambar 5.24, total yang terletak pada *data properties* telah berubah menjadi 7750000 secara otomatis. Hal ini karena total menggunakan *business logic* yang telah diimplementasikan.



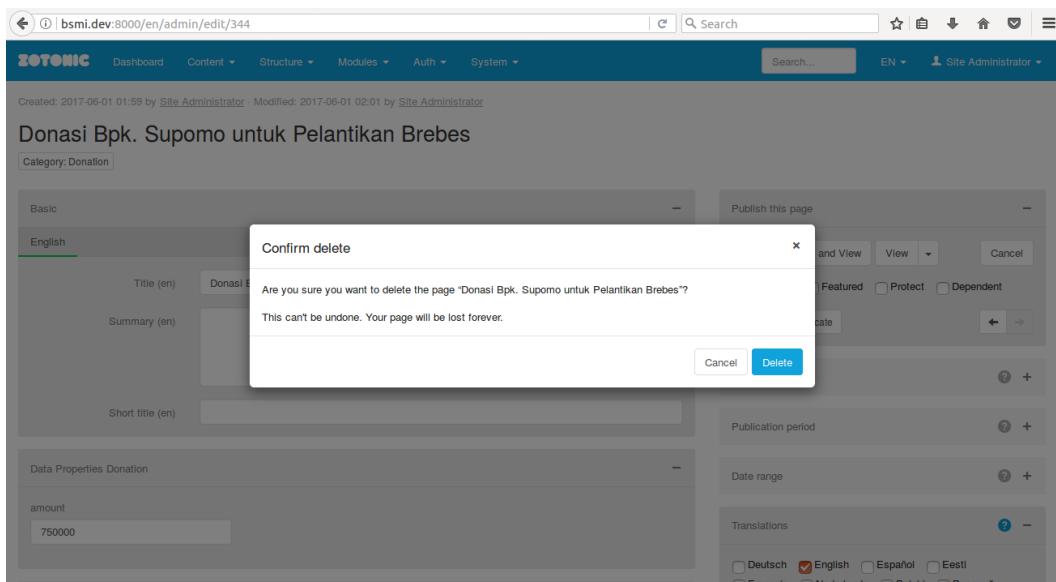
Gambar 5.24: Tampilan halaman mengubah detail *page* kategori program setelah ada donasi

Kasus pengujian berikutnya adalah menghapus donasi yang sudah ada untuk

mengecek apakah *business logic* dapat menyesuaikan dengan perubahan yang terjadi atau tidak. Setelah membuka halaman untuk mengubah detail *page* dari donasi yang ingin dihapus seperti pada Gambar 5.25, tekan tombol *delete* untuk menghapus *page*. Setelah menekan tombol *delete*, akan muncul konfirmasi apakah ingin melanjutkan penghapusan *page* atau membatalkannya. Hal ini dapat dilihat pada Gambar 5.26. Tekan tombol *delete* untuk menghapus *page* tersebut.



Gambar 5.25: Tampilan halaman mengubah detail *page* pada kategori Donasi



Gambar 5.26: Tampilan konfirmasi penghapusan *page*

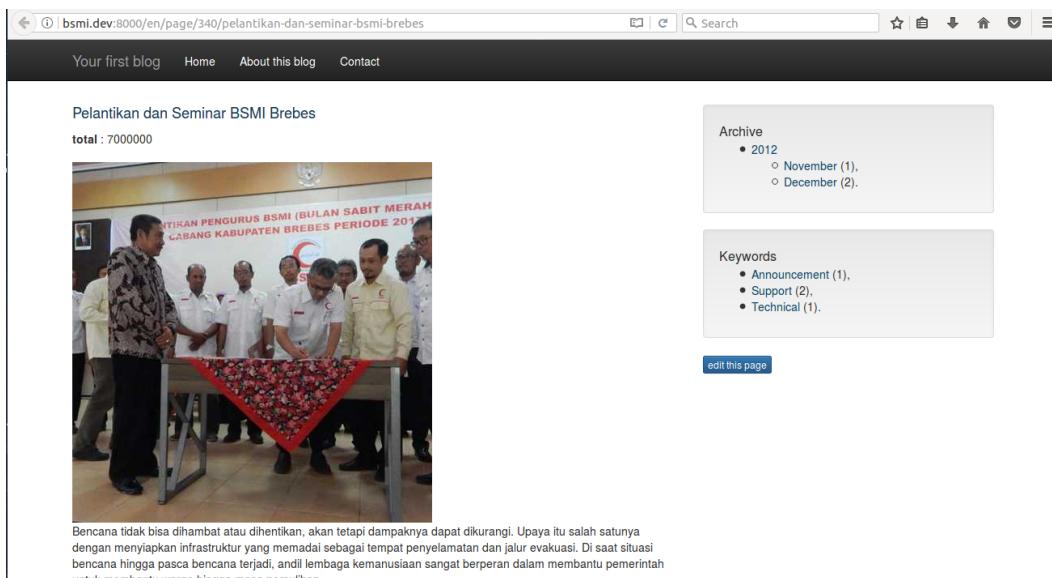
Ketika donasi telah berhasil dihapus, *adaptor* akan melakukan penghapusan donasi yang terletak pada database eksternal sesuai dengan id dari donasi yang dihapus. Hal ini dapat dilihat pada Gambar 5.27 dimana donasi dengan id 344 sudah

tidak ada lagi.

	id	name	amount	p_id
	342	Donasi Pusat untuk Pelantikan Brebes	7000000	340

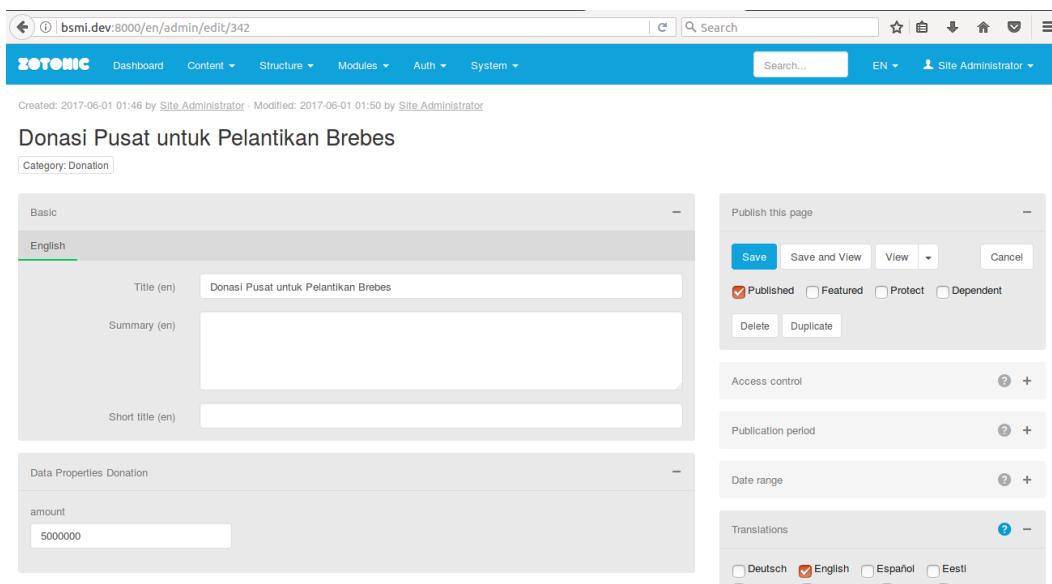
Gambar 5.27: Penghapusan tupel pada database eksternal setelah penghapusan donasi

Hal ini juga akan mengubah jumlah total donasi yang diberikan kepada program "Pelantikan dan Seminar BSMI Brebes" karena penghapusan donasi "Donasi Bpk. Supomo untuk Pelantikan Brebes". Donasi yang terhubungan dengan program tersebut hanya tinggal donasi "Donasi Pusat untuk Pelantikan Brebes" yang memiliki *amount* 7000000 sehingga total donasi yang diterima oleh program tersebut akan menjadi 7000000 juga. Untuk melihat apakah perubahannya telah terjadi atau tidak, dapat mengakses halaman dari Pelantikan dan Seminar BSMI Brebes. Pada Gambar 5.28, yang merupakan tampilan halaman Pelantikan dan Seminar BSMI Brebes setelah penghapusan donasi "Donasi Bpk. Supomo untuk Pelantikan Brebes", dapat dilihat bahwa total donasi sudah berubah menjadi 7000000



Gambar 5.28: Tampilan *page* Pelantikan dan Seminar BSMI Brebes setelah penghapusan donasi

Pengujian berikutnya adalah untuk mencoba jika suatu donasi yang sudah ada dilakukan perubahan terhadap nilai dari *amount* apakah *business logic* akan berubah juga. Untuk melakukan hal tersebut, buka halaman untuk mengubah detail dari donasi yang akan diubah. Pada kasus ini, penulis akan mengubah donasi "Donasi Pusat untuk Pelantikan Brebes". Seperti pada Gambar 5.29, penulis mengubah nilai *amount* donasi tersebut dari 7000000 menjadi 5000000. Selanjutnya tekan tombol *save* untuk menyimpan perubahan yang terjadi.



Gambar 5.29: Tampilan halaman mengubah detail *page* pada kategori Donasi

Ketika perubahan berhasil disimpan, maka perubahan tersebut akan tersimpan juga pada database eksternal. Seperti pada Gambar 5.30, terlihat bahwa donasi dengan id 342 dimana nilai dari *amount* telah berubah menjadi 5000000.

The screenshot shows the phpMyAdmin interface for the 'donation' table in the 'abs_program' database. The table has columns: id, name, amount, and p_id. One record is present: id 342, name 'Donasi Pusat untuk Pelantikan Brebes', amount 5000000, and p_id 340. The SQL query at the top is 'SELECT * FROM `donation`'.

Gambar 5.30: Perubahan database eksternal setelah perubahan donasi

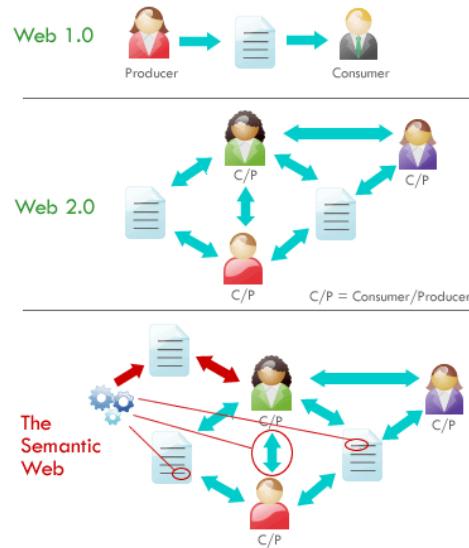
Perubahan ini juga dapat dilihat pada halaman dari Pelantikan dan Seminar BSMI Brebes seperti pada Gambar 5.31

The screenshot shows a web page titled 'Pelantikan dan Seminar BSMI Brebes - Update'. It features a photograph of several men in formal attire standing behind a long table covered with a red cloth. A banner in the background reads 'PELANTIKAN PENGURUS BSMI (BULAN SABIT MERAH) GABANG KABUPATEN BREBES PERIODE 2012-2013'. Below the photo, there is a caption in Indonesian: 'Bencana tidak bisa dihambat atau dihentikan, akan tetapi dampaknya dapat dikurangi. Upaya itu salah satunya dengan menyiapkan infrastruktur yang memadai sebagai tempat penyelamatan dan jalur evakuasi. Di saat situasi bencana hingga pasca bencana terjadi, andil lembaga kemanusiaan sangat berperan dalam membantu pemerintah untuk membantu warga hingga masa pemulihan.' On the right side, there are two sidebar boxes: 'Archive' (listing 2012, November 1, December 2) and 'Keywords' (listing Announcement 1, Support 2, Technical 1).

Gambar 5.31: Tampilan *page* Pelantikan dan Seminar BSMI Brebes setelah perubahan

5.4 Contoh Keterhubungan Data pada Web BSMI

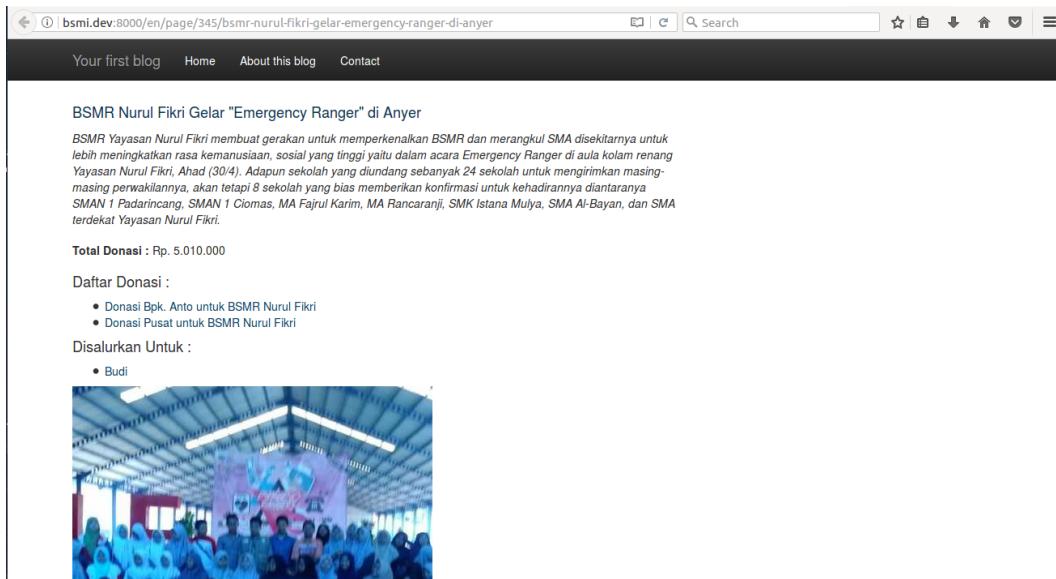
Setelah pada subbab sebelumnya telah dilakukan proses pembuatan *page* pada web BSMI, pada subbab ini akan membahas mengenai keterhubungan data dari *page-page* yang telah dibuat tersebut. Berbeda dengan *web* pada umumnya, *web* yang menerapkan konsep semantik akan dapat langsung mengolah informasi berdasarkan keterhubungan data yang dimilikinya.



Gambar 5.32: Perbedaan *web* semantik dengan *web* biasa

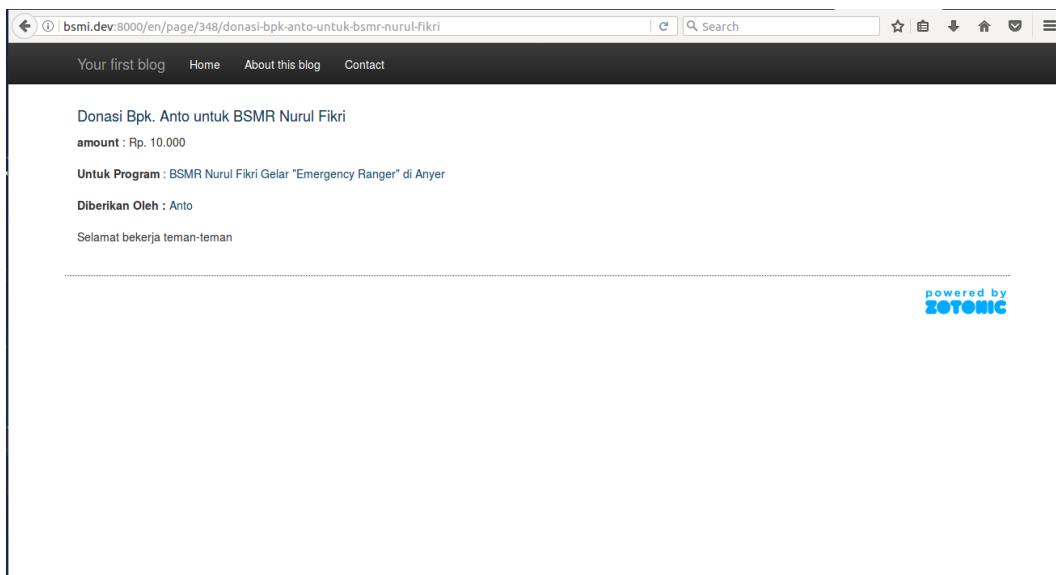
Pada Gambar 5.32, terdapat perbandingan antara *web* biasa dengan *web* semantik. Pada teknologi *web* 1.0, sebuah *page* khusus dibuat oleh *producer* untuk satu *customer*. Pada teknologi ini, *customer* hanya berperan sebagai pembaca saja. Contohnya dari penerapan teknologi ini adalah blog. Selanjutnya ditemukan teknologi *web* 2.0 dimana *customer* dapat berperan sebagai *producer* juga. Contoh dari penerapan teknologi ini adalah sosial media seperti Twitter, Facebook, dan lain-lain. Namun, pada teknologi ini, informasi-informasi yang ada hanya sebatas untuk ditampilkan saja. Sehingga muncul teknologi *web* 3.0 atau disebut *web* semantik dimana informasi-informasi yang terdapat pada *web* dapat diolah dan ditampilkan pada suatu *page*.

Zotonic sebagai sebuah *web framework* yang telah mengimplementasikan semantik dapat digunakan sebagai sebuah *engine* untuk mengolah dan menampilkan hasilnya pada sebuah *page*. Berikut merupakan hasil dari penerapan keterhubungan data pada *web* BSMI dengan menggunakan beberapa bagian dari ontologi *Charity Organization*.

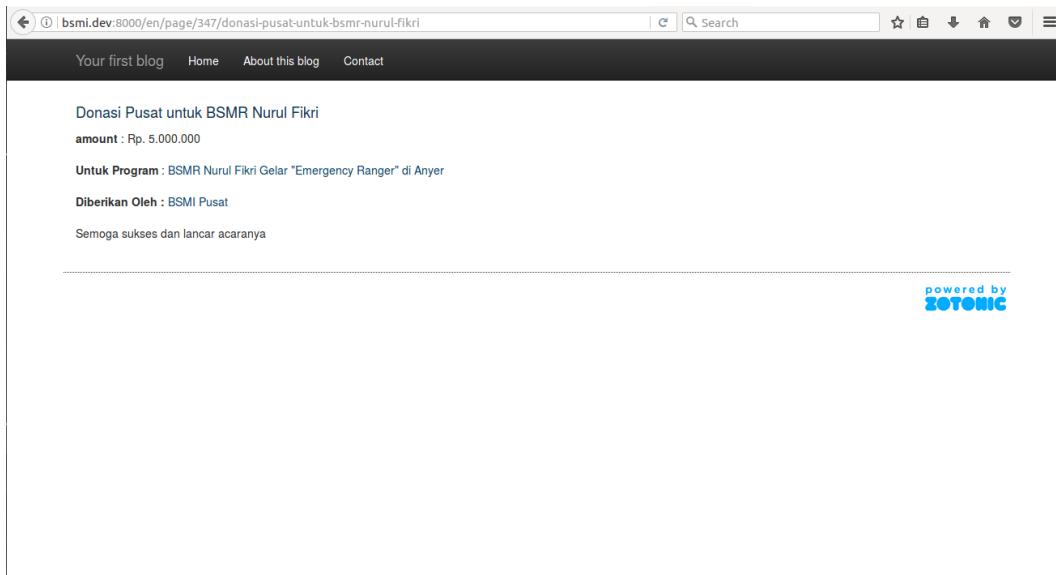


Gambar 5.33: Tampilan *page Program* yang telah menampilkan keterhubungan data

Pada Gambar 5.33, merupakan tampilan dari sebuah *page* dengan kategori *Program*. Tampilan ini menggabungkan informasi yang didapatkan dari *page Beneficiary* dan *Donation* untuk ditampilkan pada tampilan *Program*. Pada *page* ini, ditampilkan total donasi yang diterima untuk program ini yang dihitung dari *amount* seluruh donasi yang terhubung ke program ini seperti yang dijelaskan pada bab sebelumnya. Selanjutnya terdapat juga informasi daftar donasi yang diberikan kepada program ini serta informasi disalurkan kepada siapa dana dari program ini.



Gambar 5.34: Tampilan *page Donasi* yang telah menampilkan keterhubungan data

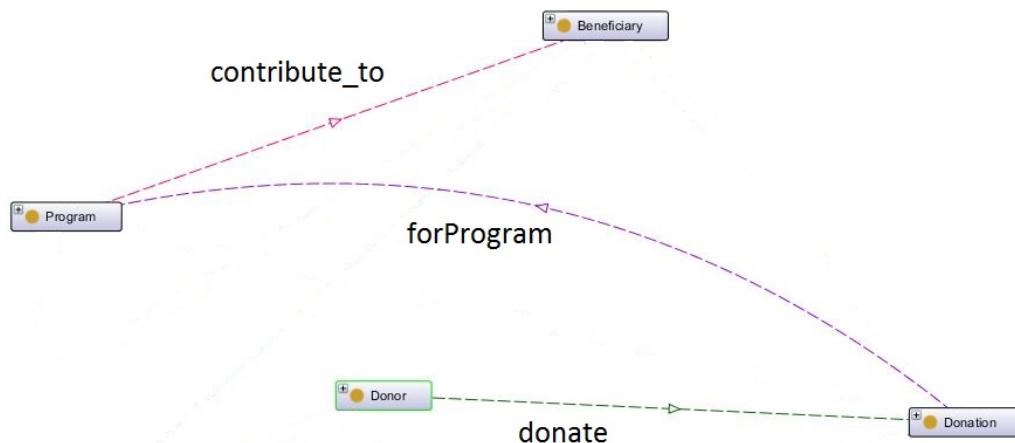


Gambar 5.35: Tampilan *page Donasi* yang telah menampilkan keterhubungan data

Pada Gambar 5.34 dan Gambar 5.35, merupakan tampilan dari sebuah *page* dengan kategori *Donation*. Tampilan tersebut menggabungkan informasi yang didapatkan dari *page Program* dan *Donor*. Pada *page* tersebut, ditampilkan informasi mengenai donasi tersebut disalurkan untuk program yang mana dan donasi tersebut diberikan oleh siapa.

5.5 Kesesuaian Keterhubungan Data dengan Ontologi

Berdasarkan hasil pada subbab 5.3 dan 5.4, dapat dilihat bahwa dapat diimplementasikan sebuah *web* yang berbasis semantik dimana informasi-informasi yang terdapat pada setiap *page* dapat diolah dan ditampilkan pada suatu *page*.



Gambar 5.36: Potongan graf dari ontologi *Charity Organization*

Pada Gambar 5.36 merupakan beberapa bagian dari ontologi *Charity Organization*. Pada subbab 5.3 merupakan proses pembuatan *page* atau *individual* dari *class* yang terdapat pada ontologi tersebut. *Page-page* tersebut memiliki informasi hubungan yang menghubungkan antara *class* yang selanjutnya digunakan untuk menampilkan keterhubungan data pada subbab 5.4.

Informasi-informasi tersebut didapatkan langsung melalui ontologinya tanpa harus melakukan pemanggilan kepada *database*. Hal ini berbeda dengan *web* pada umumnya yang harus membuat *Model*, *View* dan *Controller* (MVC) untuk dapat mengambil informasi dari database dan menampilkannya ke dalam *web*. Dengan menggunakan Zotonic, pengambilan informasi dari ontologi dan menampilkan ke dalam *page* cukup dilakukan pada *View* saja. Untuk mempermudah pengguna, Zotonic telah membuat *model* yang dapat diakses dari *template engine* sehingga pengguna dapat mengambil informasi terkait hubungan antara *page* juga.

Hal ini tentu akan mempermudah pengguna untuk dapat mengembangkan *web* semantik menggunakan Zotonic. Karena untuk mengubah pada sisi tampilan, hal tersebut dapat dilakukan oleh pengguna yang tidak memiliki kemampuan ataupun latar belakang ilmu komputer.

BAB 6

PENUTUP

Pada bab ini, akan dijelaskan kesimpulan yang didapatkan berdasarkan hasil penelitian serta saran untuk pengembangan program kedepannya.

6.1 Kesimpulan

Pengembangan program untuk melakukan integrasi ontologi dan *web services* dapat dilakukan dengan memanfaatkan sebuah *adaptor* yang dibuat pada Zotonic dimana *adaptor* tersebut yang akan menghubungkan keduanya seperti yang dilakukan pada penelitian ini. *Adaptor* tersebut akan membaca file rules.txt sebagai tabel *mapping* untuk proses pemanggilan *web services*. Hasil dari implementasi *adaptor* ini dapat digunakan baik melalui *template engine* maupun model lainnya. *Adaptor* tersebut terdiri dari beberapa fungsi utama seperti fungsi untuk pembacaan tabel *mapping*, pemanggilan *web service* hingga fungsi untuk validasi parameter.

Penggunaan *adaptor* pada *template engine* dapat dimanfaatkan untuk mengantikan pembuatan *business logic* yang masih harus dibuat secara manual pada kode. Hal ini berguna agar *business logic* dapat bersifat lebih dinamis dan dapat disesuaikan dengan kebutuhan. Sehingga dapat dihasilkan beberapa web yang memiliki struktur yang sama tetapi memiliki *business logic* yang berbeda dengan memanfaatkan *ABS microservices*. Penggunaan *adaptor* pada *model* lainnya dapat digunakan salah satunya untuk mengirim *resource* dari zotonic ke *web services*. Hal ini bertujuan agar data yang terdapat beberapa *site* zotonic nantinya dapat disimpan pada suatu database eksternal sehingga dapat dilakukan analisis terhadap data-data tersebut.

Zotonic juga menyediakan *model* yang dapat digunakan untuk mengolah informasi antar *page* dan menampilkannya pada *page*. Sehingga berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa integrasi ontologi dan *web service* pada zotonic telah tercapai. Hal ini terbukti melalui studi kasus dimana dapat dihasilkan sebuah *web* yang menampilkan hasil *business logic* yang memanfaatkan *adaptor* serta memanfaatkan informasi dari *page* lainnya berdasarkan ontologi untuk dapat ditampilkan pada *page* yang inginkan.

6.2 Saran

Setelah melakukan penelitian integrasi ontologi dan *web services* pada ontologi, terdapat beberapa saran yang dapat dilakukan untuk proses pengembangan selanjutnya. Berikut adalah saran untuk proses pengembangan selanjutnya:

1. Melakukan pemetaan antara database yang digunakan oleh zotonic dan database eksternal. Hal ini tidak dilakukan pada penelitian ini karena database yang digunakan oleh zotonic sendiri masih bisa terjadi perubahan kedepannya.
2. Melakukan penambahan validasi pada model m_abs sehingga dapat lebih mudah mengetahui penyebab terjadinya *error*. Pada penelitian ini, validasi yang dilakukan pada model m_abs hanya sebatas pengecekan jumlah parameter antara parameter yang diberikan pada *json* dan jumlah parameter yang diberikan pada tabel *mapping*.
3. Pembuatan tabel *mapping* secara otomatis dari ontologi dan bentuk tabel *mapping* yang lebih sesuai dengan informasi yang didapatkan dari ontologi karena pada penelitian ini, pembuatan tabel *mapping* masih dibuat secara manual tanpa melihat informasi yang dapat diterima dari ontologi.
4. Melakukan *migrasi* pada Zotonic ketika adanya perubahan struktur dari ontologi agar data dari *site* yang lama tetap dapat digunakan saat *rebuild* Zotonic.

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].
- Berners-Lee, T., Hendler, J., dan Lassila, O. (2001). The semantic web. Scientific American.
- 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].
- Pangukir, B. T. (2016). Pembentukan otomatis aplikasi berbasis web dengan makanan berupa ontologi: Studi kasus web bsmi. Depok: Universitas Indonesia.
- Pohl, K., Bäckle, G., dan van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer.
- Schoop, M., de Moor, A., dan Dietz, J. L. (2006). The pragmatic web: A manifesto. Communications of The ACM.
- 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

Berkas ini ditaruh pada `src/models/`. Berkas ini merupakan *adaptor* yang akan menghubungkan antara ontologi dan *web services*

Kode 1: Berkas 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

Berkas ini ditaruh pada direktori *root* dari zotonic. Berkas ini berfungsi sebagai tabel *mapping*.

Kode 2: Berkas rules.txt

```
{  
    "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],  
    "checkExistProgram" : ["http://54.169.128.6:8080/abs/program/  
        checkExist", 1],  
    "checkExistDonation" : ["http://54.169.128.6:8080/abs/donation/  
        checkExist", 1]  
}
```

LAMPIRAN 3 : STRUKTUR ZOTONIC

Berikut adalah struktur dari Zotonic setelah implementasi penambahan berkas yang digunakan sebagai *adaptor*.

```
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
rules.txt

LAMPIRAN 4 : KODE SUMBER WEB SERVICE

Web Services ini digunakan untuk melakukan ujicoba terhadap *adaptor* yang telah diimplementasikan. *Web Services* ini hanya bersifat sementara karena belum adanya *web service* dari ABS *microservices* untuk ontologi yang digunakan.

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);
    }
}
```