UNIVERSITAS INDONESIA

PENGEMBANGAN PROGRAM INTEGRASI ONTOLOGI DAN *WEB SERVICES* PADA ZOTONIC

SKRIPSI

ANDRI KURNIAWAN
1306382064

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNI 2017

UNIVERSITAS INDONESIA

PENGEMBANGAN PROGRAM 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	:	Pengembangan Program 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

Alhamdulillahirabbil'alamin, segala puji dan syukur kehadirat Tuhan Yang Maha Esa, Allah Subhana Huwataala, karena hanya dengan hidayah dan rahmat-Nya, penulis dapat menyelesaikan pembuatan skripsi ini.

Allahumma sholli 'alaa sayyidina Muhammad, Sholawat serta salam tak henti-hentinya dipanjatkan kepada Rasulullah SAW, atas peranannya di muka bumi dalam memberikan tuntunan kepada seluruh umat manusia, dan sebagai inspirasi atas seluruh manusia sebagai manusia dengan akhlak terbaik.

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

Depok, 17 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:

Pengembangan Program Integrasi Ontologi dan *Web Services* pada Zotonic

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

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 5 Juni 2017

Yang menyatakan

(Andri Kurniawan)

ABSTRAK

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

Abstrak INA

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

ABSTRACT

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

Abstract in Eng

Keywords:
one,two,three

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Menurut **?** terdapat 3 buah contoh untuk membuat enumerate pada latex (**?**):

1. Makan
2. Minum

Menurut **?**, pemodelan yang sama apabila dijalankan dengan komputer *Dual Core* maka akan membutuhkan waktu 1 tahun dengan asumsi memori yang dibutuhkan cukup (**?**).

1.2 Perumusan Masalah

Pada bagian ini akan dijelaskan mengenai definisi permasalahan yang dihadapi dan ingin diselesaikan serta asumsi dan batasan yang digunakan dalam menyelesaikannya.

1.3 Tujuan dan Manfaat Penelitian

Dibawah ini adalah contoh itemize :

- Terimplementasinya .
- Menyelesaikan masalah .

1.4 Tahapan Penelitian

@todo
Tuliskan tujuan penelitian.

1.5 Ruang Lingkup Penelitian

1.6 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN
- Bab 2 TINJAUAN PUSTAKA
- Bab 3 RANCANGAN
- Bab 4 IMPLEMENTASI
- Bab 5 HASIL
- Bab 6 PENUTUP

@todo

Tambahkan penjelasan singkat mengenai isi masing-masing bab.

BAB 2
TINJAUAN PUSTAKA

Bab ini berisi .... Pada sub-bab 2.1 akan dijelaskan dasar-dasar ...

2.1 X si sesuatu

Dokumen LaTeX sangat mudah, seperti halnya membuat dokumen teks biasa. Ada beberapa perintah yang diawali dengan tanda '\'. Seperti perintah \\ yang digunakan untuk memberi baris baru. Perintah tersebut juga sama dengan perintah \newline. Pada bagian ini akan sedikit dijelaskan cara manipulasi teks dan perintah-perintah LaTeX yang mungkin akan sering digunakan. Jika ingin belajar hal-hal dasar mengenai LaTeX, silahkan kunjungi:

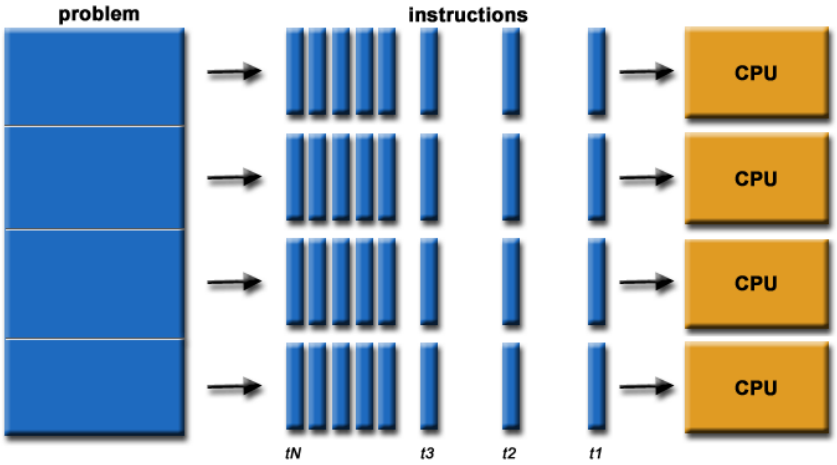
- http://frodo.elon.edu/tutorial/tutorial/, atau
- http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/

2.1.1 Pengertian X

Setiap gambar dapat diberikan caption dan diberikan label. Label dapat digunakan untuk menunjuk gambar tertentu. Jika posisi gambar berubah, maka nomor gambar juga akan diubah secara otomatis. Begitu juga dengan seluruh referensi yang menunjuk pada gambar tersebut.

Contoh sederhana adalah Gambar 2.1. Silahkan lihat code LaTeX dengan nama bab2-landasan-teori.tex untuk melihat kode lengkapnya. Harap diingat bahwa caption untuk gambar selalu terletak dibawah gambar.

Dibawah adda figure, jangn lupa dimention dengan 2.1.



Gambar 2.1: Contoh masalah yang dikerjakan secara paralel

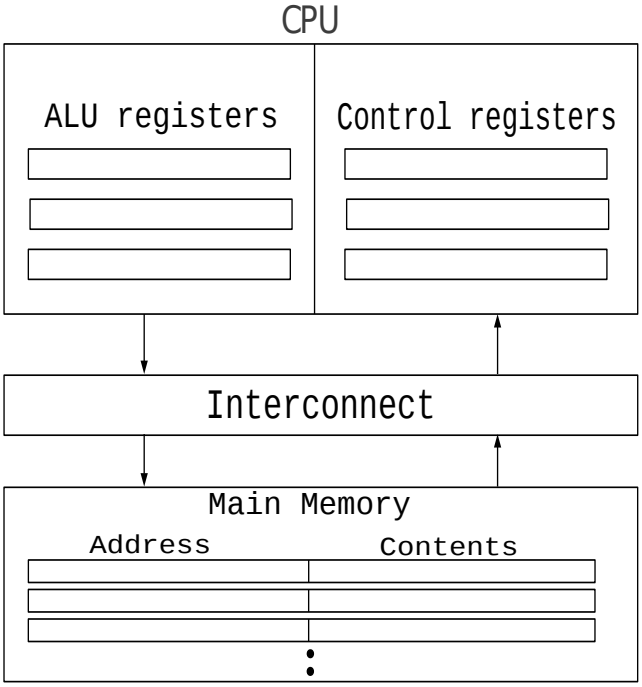
Sumber gambar: (**?**)

2.1.2 Klasifikasi X

Figure dalam enum dan dua sitasi sekaligus (**??**) :

1. ***Bold Italic***

   Penjelasan..... Untuk gambarannya dapat dilihat di Gambar 2.2.



Gambar 2.2: Arsitektur klasik *von Neumann*
Sumber gambar terinspirasi dari: (**?**)

2. ***Sesuatu banget***

   Penjelasan.....

2.2 *Section in Eng*

Hal pertama yang mungkin ditanyakan adalah bagaimana membuat huruf tercetak tebal, miring, atau memiliki garis bawah. Pada Texmaker, Anda bisa melakukan hal ini seperti halnya saat mengubah dokumen dengan LO Writer. Namun jika tetap masih tertarik dengan cara lain, ini dia:

- **Bold**
  Gunakan perintah \textbf{} atau \bo{}.
- *Italic*
  Gunakan perintah \textit{} atau \f{}.
- Underline
  Gunakan perintah \underline{}.
- $\overline{Overline}$
  Gunakan perintah \overline.
- ^{superscript}
  Gunakan perintah \^{}.
- $_{subscript}$
  Gunakan perintah _{}.

Perintah \f dan \bo hanya dapat digunakan jika package uithesis digunakan.

2.2.1 Pengertian *Section in Eng*

2.2.2 Next Subsection *Section in Eng*

2.3 *Keatas lagi*

Contoh cite yang ga ada **?**. Cite author **?**,cite tahun **?**, cite mention **?**, dan cite di akhir kalimat (**?**).

2.3.1 *Masuk lagi*

Footnote example nih : MPICH [1], LAM/MPI [2], dan OpenMPI [3] (**?**). MPI-3 sedang dalam tahap perencanaan [4]. Fungsi-fungsi tersebut berada di tabel 2.1. (Contoh tabel).

Tabel 2.1: Fungsi fundamental MPI

No.	Nama Fungsi	Penjelasan
1	MPI_Init	Memulai kode MPI
2	MPI_Finalize	Mengakhiri kode MPI
3	MPI_Comm_size	Menentukan jumlah proses
4	MPI_Comm_rank	Menentukan label proses
5	MPI_Send	Mengirim pesan
6	MPI_Recv	Menerima pesan

Sumber tabel: taro sitasi disini, if i were u

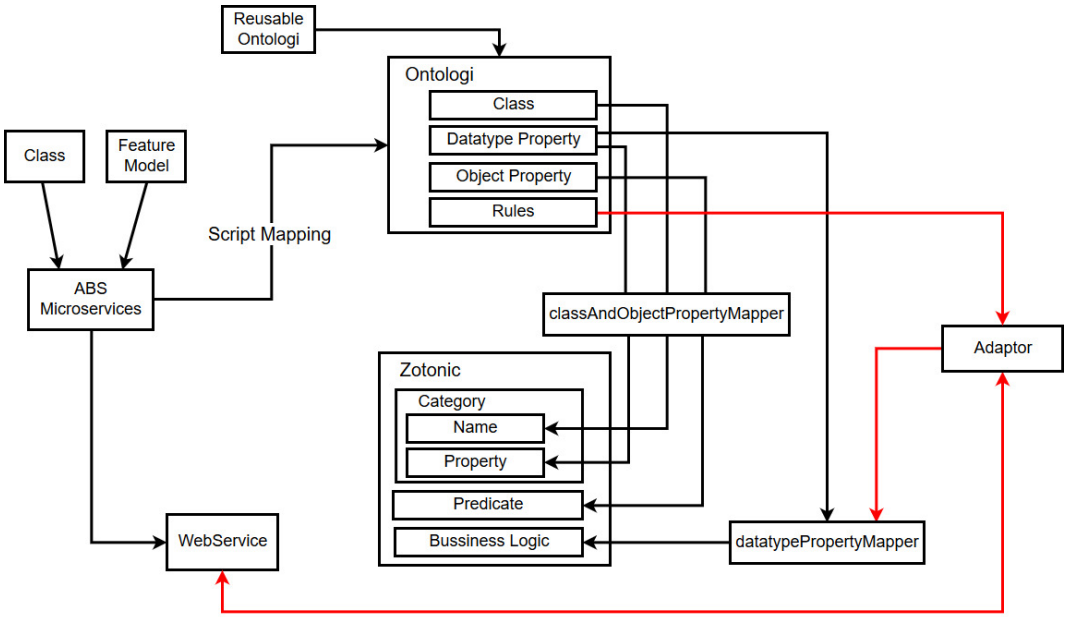
[1] http://www.mpich.org/
[2] http://www.lam-mpi.org/
[3] www.open-mpi.org
[4] http://meetings.mpi-forum.org/MPI_3.0_main_page.php

BAB 3
RANCANGAN

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

3.1 Rancangan Integrasi Ontologi dan *Web Service*

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



Gambar 3.1: Rancangan integrasi ontologi dan web service

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

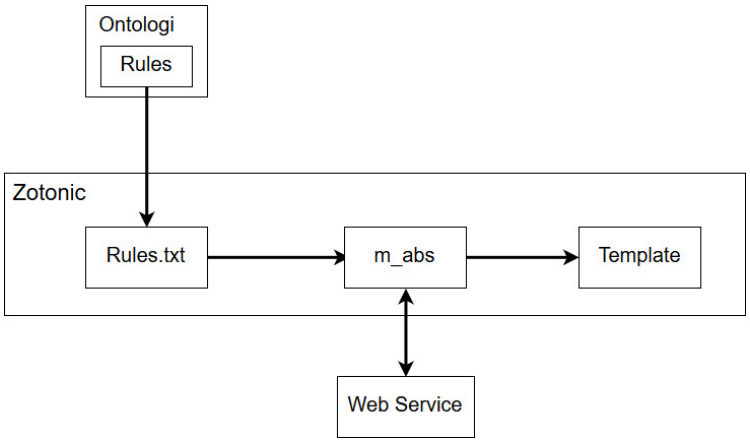
Menggunakan ontologi yang telah dirancang sebelumnya, ontologi tersebut akan dipetakan ke dalam struktur dari zotonic yang akan digunakan. Proses pemetaan dari ontologi ke dalam zotonic ini sendiri telah dilakukan pada penelitian sebelumnya oleh Bravyto dimana setiap *class* dari ontologi akan dipetakan 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 zotonic 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.

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

3.2 Desain *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 gam-

bar berikut ini.



Gambar 3.2: Rancangan Adaptor

Seperti yang dapat dilihat pada Gambar 3.2, *rules* yang terdapat pada ontologi akan dilakukan pemetaan menjadi tabel *rules* yang akan disimpan pada file rules.txt seperti yang terdapat pada kode 3.1. Namun, proses pemetaan ini berada diluar dari topik penelitian penulis sehingga untuk keperluan penelitian ini maka penulis membuat sebuah tabel *rules* untuk mengganti proses pemetaan tersebut.

Kode 3.1: Contoh tabel *rules*

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

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

BAB 4
IMPLEMENTASI

4.1 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 4.1: Struktur tabel *rules*

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

Setelah tabel rules selesai dibuat sesuai dengan struktur pada kode 4.1, tabel rules tersebut disimpan ke dalam sebuah file yang bernama rules.txt dan ditaruh pada *root* dari zotonic. Untuk dapat menjalankan fungsi *adaptor* yang diinginkan, penulis membuat sebuah model baru pada zotonic dengan nama model abs. Untuk membuat sebuah model pada zotonic, zotonic mengharuskan setiap model untuk mengekspor beberapa fungsi yang dimiliki oleh zotonic yaitu fungsi model find value, model to list, dan model value agar fungsi tersebut dapat digunakan melalui *template engine*.

Kode 4.2: Fungsi yang harus diekspor untuk model

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

Seperti yang dapat dilihat pada kode 4.2, agar fungsi tersebut dapat dijalankan pada *template engine*, tentu harus diimplementasi sesuai dengan kebutuhan. Sesuai dengan kebutuhannya agar *adaptor* dapat dipanggil melalui *template engine*, maka perlu didefinisikan terlebih dahulu bagaimana nantinya *adaptor* dipanggil. Pada penelitian ini, penulis mendefinisikan untuk pemanggilan *adaptor* pada *template engine* dilakukan dengan membuat perintah `m.abs.namaFungsi[{query param=value}]`

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 4.3: 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 4.3, tahap awal untuk mengimplementasikan fungsi m_find_value adalah membuat sebuah *specifications* untuk fungsi 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.

Ketika model abs dijalankan pada sisi *template engine*, model abs akan melakukan pencocokan *pola* terhadap pemanggilan model abs pada *template engine*. Untuk melakukan pencocokan pola ini, penulis memanfaatkan fungsi model find value yang telah disediakan oleh zotonic.

Kode 4.4: Implementasi fungsi m to list

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

Kode 4.5: Implementasi fungsi m value

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

Seperti pada gambar, tabel juga dapat diberi label dan caption. Caption pada tabel terletak pada bagian atas tabel. Contoh tabel sederhana dapat dilihat pada Tabel 4.1.

Tabel 4.1: Contoh Tabel

	kol 1	kol 2
baris 1	1	2
baris 2	3	4
baris 3	5	6
jumlah	9	12

Ada jenis tabel lain yang dapat dibuat dengan LaTeX berikut beberapa diantaranya. Contoh-contoh ini bersumber dari http://en.wikibooks.org/wiki/LaTeX/Tables

Tabel 4.2: An Example of Rows Spanning Multiple Columns

No	Name	Week 1			Week 2		
		A	B	C	A	B	C
1	Lala	1	2	3	4	5	6
2	Lili	1	2	3	4	5	6
3	Lulu	1	2	3	4	5	6

Tabel 4.3: An Example of Columns Spanning Multiple Rows

Percobaan	Iterasi	Waktu
Pertama	1	0.1 sec
Kedua	1	0.1 sec
	3	0.15 sec
Ketiga	1	0.09 sec
	2	0.16 sec
	3	0.21 sec

Tabel 4.4: An Example of Spanning in Both Directions Simultaneously

		Title			
		A	B	C	D
Type	X	1	2	3	4
	Y	0.5	1.0	1.5	2.0
Resource	I	10	20	30	40
	J	5	10	15	20

4.2 Implementasi *Refactoring Business Logic*

Berkas ini berisi seluruh berkas Latex yang dibaca, jadi bisa dikatakan sebagai berkas utama. Dari berkas ini kita dapat mengatur bab apa saja yang ingin kita tampilkan dalam dokumen.

4.3 Implementasi Pemanggilan *Web Service*

Berkas ini berguna untuk mempermudah pembuatan beberapa template standar. Anda diminta untuk menuliskan judul laporan, nama, npm, dan hal-hal lain yang dibutuhkan untuk pembuatan template.

4.4 Implementasi *Rules*

Berkas istilah digunakan untuk mencatat istilah-istilah yang digunakan. Fungsinya hanya untuk memudahkan penulisan. Pada beberapa kasus, ada kata-kata yang harus selalu muncul dengan tercetak miring atau tercetak tebal. Dengan menjadikan kata-kata tersebut sebagai sebuah perintah LaTeX tentu akan mempercepat dan mempermudah pengerjaan laporan.

BAB 5
HASIL

5.1 Implementasi *Cluster*

5.1.1 Instalasi *Frontend*

Tabel model lain, ditunjukkan pada tabel 5.1.

Tabel 5.1: Informasi *cluster* X

Host Name	X
Cluster Name	X
Certificate Organization	UI
Certificate Locality	Depok
Certificate State	West Java
Certificate Country	ID
Contact	X
URL	http://grid.ui.ac.id

Ada pagebreak disini.

Another type of table

Tabel 5.2: Perbandingan Partisi *default* dan manual

	Partisi default	Partisi manual yang dilakukan
/	16 GB	30 GB
/var	4 GB	18 GB
swap	1 GB	2 GB
/export	55 GB	26 GB

Program menghasilkan keluaran seperti pada kode 5.1.

Kode 5.1: Keluaran output

```
[root@nas-0-0 ~]# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sda4[0] sdb2[1]
      1917672312 blocks super 1.2 [2/2] [UU]

unused devices: <none>
[root@nas-0-0 ~]# mdadm --detail /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Fri May  3 15:38:52 2013
     Raid Level : raid1
     Array Size : 1917672312 (1828.83 GiB 1963.70 GB)
  Used Dev Size : 1917672312 (1828.83 GiB 1963.70 GB)
   Raid Devices : 2
  Total Devices : 2
    Persistence : Superblock is persistent

    Update Time : Tue May 28 11:27:49 2013
          State : clean
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

           Name : nas-0-0.local:0  (local to host nas-0-0.local)
           UUID : 0754726d:3dfbd4b9:42b0f587:68631556
         Events : 28

    Number   Major   Minor   RaidDevice State
       0       8        4        0      active sync   /dev/sda4
       1       8       18        1      active sync   /dev/sdb2
```

5.1.2 Konfigurasi

Contoh verbatim dalam itemize :

- **Bold ini**
  dijalankan perintah berikut :

```
# javac Ganteng.java
# java Ganteng
```

  Perilaku sistem

```
# hai
# enable
# cd /export/rocks/install/
# create distro
# sh sesuatu.sh
# reboot
```

- **Menambahkan *package* pada *compute node***
  Langkah yang dilakukan adalah sebagai berikut :

  1. Masuk ke dalam direktori `/procfs/`
  2. Membuat/Mengubah berkas `xx.xml`. Jika tidak terdapat berkas tersebut, dapat disalin dari `skeleton.xml`.
  3. Menambahkan *package* yang ingin dipasang pada *compute node* diantara *tag* `<package>` seperti berikut : `<package>[package yang akan dipasang]</package>`.
  4. Menjalankan perintah berikut termasuk perintah untuk melakukan instalasi ulang seluruh *compute node*:

```
# cd /export/somedir
# create
# run host
```

5.1.2.1 semakin ke dalam

Kode 5.2: Keluaran mentah untuk detail *job*

```
[ardhi@xx ~]$ qstat -f 138
Job Id: 138.xx
    Job_Name = cur-1000-1np
    Job_Owner = ardhi@xx
    resources_used.cput = 27:21:35
    resources_used.mem = 86060kb
    resources_used.vmem = 170440kb
    resources_used.walltime = 27:24:50
    job_state = R
    queue = default
    server = hastinapura.grid.ui.ac.id
    Checkpoint = u
    ctime = Fri May 31 10:27:37 2013
    Error_Path = xx:/home/ardhi/xx/curcumin-1000/cur-1000-1np.e138
    exec_host = compute-0-5/0
    exec_port = 15003
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = e
    Mail_Users = ardhi.putra@ui.ac.id
    mtime = Fri May 31 10:27:47 2013
    Output_Path = xx:/home/ardhi/xx/curcumin-1000/cur-1000-1np.o138
    Priority = 0
    qtime = Fri May 31 10:27:37 2013
    Rerunable = True
    Resource_List.nodes = 1:ppn=1
    session_id = 5768
    etime = Fri May 31 10:27:37 2013
    submit_args = cur-1000-1np.pbs
    start_time = Fri May 31 10:27:47 2013
    submit_host = xx
    init_work_dir = /home/ardhi/xx/curcumin-1000
```

5.2 Pengujian

5.2.1 Kasus Uji

Berwarna!

Kode 5.3: Potongan skrip submisi *job* melalui torqace

```
# Go To working directory
cd $PBS_O_WORKDIR

#openMPI prerequisite
. /opt/torque/etc/openmpi-setup.sh
```

```
mpirun -np 5 -machinefile $PBS_NODEFILE mdrun -v -s \
  curcum400ps.tpr -o md_prod_curcum400_5np.trr -c lox_pr.gro
...
```

5.2.2 Kasus Uji

Contoh skrip yang dimasukkan pada *form* yang disediakan dapat dilihat pada kode 5.4.

Kode 5.4: Potongan `Makefile` *project*

```
# Make file for MPI
SHELL=/bin/sh

# Compiler to use
# You may need to change CC to something like CC=mpiCC
# openmpi : mpiCC
# mpich2  : /opt/mpich2/gnu/bin/mpicxx
CC=mpiCC
...
...
```

BAB 6
PENUTUP

Pada bab terakhir ini,

6.1 Kesimpulan

6.2 Saran

LAMPIRAN

LAMPIRAN 1 : KODE SUMBER MODEL ABS

mabs.erl

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

Kode 1: Skrip menambahkan pengguna baru

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

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

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

-include_lib("zotonic.hrl").

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

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

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

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

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

m_to_list(_, _Context) ->
  [].

m_value(_, _Context) ->
  undefined.

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

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

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

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

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

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

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

LAMPIRAN 2 : KODE SUMBER RULES

rules.txt

Kode 2: Berkas `compute.xml`

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

LAMPIRAN 8 : UAT DAN KUESIONER

Tabel 1: Tabel UAT dan Kuesioner

No.	Langkah Penggunaan	Fitur Berjalan	Tingkat Kemudahan (1-5)	Tingkat Kepuasan (1-5)	Saran / Komentar
		Berhasil /Tidak	1:Sangat sulit ; 5:sangat mudah	1 : Sangat kecewa ; 5 : sangat puas	
Use Case : Login					
1.1	Pengguna berada pada halaman depan torqace				
1.2	Pengguna memasukkan username dan password pada field yang telah disedi-akan.Kemudian menekan tombol 'login'				
1.3	Apabila Sukses, maka pengguna masuk ke dalam sistem dan dihadapkan pada menu utama				
Use Case : Register					
2.1	Pengguna berada pada halaman registrasi pengguna torqace				

2.2	Pengguna memasukkan user-name,password, dan email pada field yang telah disediakan. Kemudian menekan tombol 'submit'				
2.3	Sistem akan mengonfirmasi masukan, dan akan mengirimkan email untuk mem-beritahu pengguna apabila proses pendaf-taran telah selesai				
Use Case : Logout					
3.1	Pengguna memilih menu untuk melakukan logout				
3.2	Sistem akan mengeluarkan pengguna, dan pengguna tidak dapat menggunakan fitur-fitur utama aplikasi				
Use Case : Upload Job Sederhana					
4.1	Pengguna memilih menu upload file/pro-ject pada menu utama				
4.2	Pengguna memilih pilihan 'single file' pada tipe project				

4.3	Pengguna memilih berkas yang akan diunggah, mengisi label, dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
4.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
4.5	Sistem akan menampilkan informasi terkait berkas yang diupload				
Use Case : Upload Job Compressed					
5.1	Pengguna memilih menu upload file/project pada menu utama				
5.2	Pengguna memilih pilihan 'compressed files' pada tipe project				
5.3	Pengguna memilih arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
5.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				

5.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				
Use Case : Upload Array Job					
6.1	Pengguna memilih menu upload file/project pada menu utama				
6.2	Pengguna memilih pilihan 'array' pada tipe project				
6.3	Pengguna memilih arsip-arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
6.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
6.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				

Use Case : Melihat antrian pada queue					
7.1	Pengguna memilih menu queue status pada menu utama				
7.2	Pengguna berada pada halaman yang berisi informasi queue				
Use Case : Melihat detil antrian					
8.1	Dari halaman status queue, pengguna memilih job tertentu				
8.2	Informasi mengenai detil job tersebut ditampilkan dalam bentuk tabel				
8.2.1	Apabila job tersebut bukan milik pengguna, maka sistem akan melarang pengguna melihat informasi detil suatu job				
Use Case : Membuat script job					
9.1	Pengguna memilih untuk melakukan 'generate script' baik dari laporan upload berkas, atau dari penjelajahan direktori				
9.2	Pengguna mengisi nama job, parameter job, dan script yang akan dijalankan.				
9.3	Pengguna mengonfirmasi konfirmasi submit job				

9.4	Pengguna dapat melihat informasi script secara keseluruhan dan pesan apakah terjadi kegagalan atau tidak, serta id job yang diberikan				
Use Case : Load spesifikasi job lain					
10.1	Pengguna berada pada halaman untuk membuat script				
10.2	Pengguna memilih 'Load a Previous Job'				
10.3	Pengguna memilih job mana yang akan dimuat dan menekan tombol 'Load'				
10.4	Pengguna kembali ke halaman pembuatan script dengan spesifikasi job sebelumnya				
Use Case : Menjelajah Direktori					
11.1	Pengguna memilih menu 'View File/Project' pada menu utama				
11.2	Pengguna dapat melakukan navigasi untuk masuk ke dalam direktori tertentu, atau kembali ke direktori diatasnya, dan dapat melihat terdapat berkas apa saja dalam direktori				

Use Case : Menghapus Berkas/Direktori					
12.1	Pengguna berada pada halaman penjelajahan direktori				
12.2	Pengguna memilih pilihan untuk menghapus berkas/direktori di samping item yang akan dihapus				
12.3	Pengguna mengonfirmasi konfirmasi penghapusan				
Use Case : Mengunduh Berkas/Direktori					
13.1	Pengguna berada pada halaman penjelajahan direktori				
13.2	Pengguna memilih pilihan untuk mengunduh berkas/direktori di samping item yang akan dihapus				
Use Case : Melihat Berkas					
14.1	Pengguna berada pada halaman penjelajahan direktori				
14.2	Pengguna memilih berkas yang berupa berkas teks				
14.3	Sistem akan menampilkan konten dari berkas tersebut				