Scientific Journal of Informatics
Vol. 3, No. 1, Mei 2016
p-ISSN 2407-7658 http://journal.unnes.ac.id/nju/index.php/sji e-ISSN 2460-0040

Analisis Arsitektur Aplikasi *Web* Menggunakan *Model View Controller* (MVC) pada *Framework Java Server Faces*

Gunawan[1], Armin Lawi[2], Adnan[3]

[1]STMIK Handayani UIN Alauddin Makassar, Makassar, [2]Jurusan Matematika, Universitas Hasanuddin, Makassar,[3]Prodi Teknik ElektroUniversitas Hasanuddin, Makassar
Email: [1]gunawan@uin-alauddin.ac.id,[2]armin@unhas.ac.id, [3]adnan@unhas.ac.id

Abstrak

Aplikasi *web* yang khususnya memiliki kompleksitas besar dalam melakukan transaksi data sehingga konsep arsitektur (*pattern*) perlu menjadi perhatian khusus untuk dapat mengoptimalkan kinerja performansi sistem ketika pengguna (*user*) menggunakan dalam waktu yang bersamaan dengan jumlah yang banyak. Analisis performa arsitektur aplikasi *web* yang menggunakan model 2 (MVC) dengan menggunakan *framework Java Server Faces* (JSF) dan model 1 sebagai pembanding. Metode yang digunakan adalah *Load dan Scalability Testing* dengan dua cara yaitu uji coba terhadap *response time* karena peningkatan ukuran dari *database* dan uji coba terhadap *response time* karena peningkatan jumlah *user* yang menggunakan sistem secara bersamaan (*concurrent users*) dan waktu tunggu (*ramp-up*) yang ditentukan menggunakan Apache Jmeter. Analisis menunjukkan bahwa dalam implementasi arsitektur *web* yang menggunakan model 1 waktu rata-rata yang dibutuhkan untuk merespon permintaan *user* lebih cepat dan efisien dibanding model 2 (MVC).

Kata kunci: MVC, *Load and scalability*, *Java server faces*, Jmeter

1. PENDAHULUAN

Tren teknologi aplikasi berbasis *web* mengalami perubahan yang sangat signifikan pada proses bisnis ataupun presentasi. Aplikasi *web* yang khususnya memiliki kompleksitas besar dalam melakukan transaksi data dan konsep arsitektur (*pattern*) perlu menjadi perhatian khusus untuk dapat mengoptimalkan kinerja performansi sistem ketika pengguna (*user*) menggunakan dalam waktu yang bersamaan dengan jumlah yang banyak. Konsep arsitektur yang paling populer saat ini adalah arsitektur Model 1dan arsitektur Model 2 (MVC).

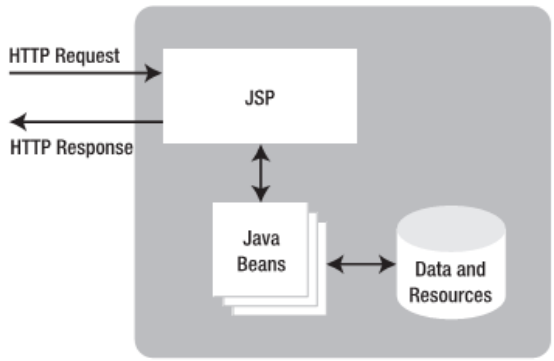
Arsitektur Model 1 logika bisnis (*business-logic)* dan presentasi (*view*) digabungkan dalam satu komponen *web*. Hal ini tidak menjadi masalah jika logika bisnis dan presentasi masih relatif sederhana. Namun jika logika menjadi semakin kompleks, pendekatan ini akan menjadi masalah dimana jika proses bisnis dan presentasi dibangun menjadi satu dapat menyulitkan para pengembang (*developer*) ketika melakukan perubahan presentasi atau proses bisnis karena ketergantungan yang tinggi antara presentasi dan proses bisnis. Arsitektur Model 2 (MVC) merupakan konsep pada beberapa *framework* aplikasi *web* dimana memisahkan pengembangan aplikasi berdasarkan komponen utama seperti manipulasi data, *user interface* dan bagian yang menjadi kontrol aplikasi. Dengan MVC aplikasi *web* dapat dikembangkan sesuai dengan kemampuan pengembang, misalnya *programmer* yang menangani bagian model dan *controller*, sedangkan *designer* yang menangani bagian *view*, sehingga dapat meningkatkan *maintainability* dan organisasi kode yang lebih mudah.

Ada banyak jenis *framework* yang dapat digunakan untuk mengimplementasikan MVC diantaranya yaitu *Java Server Faces* (JSF), *Struts*, *Seam*, *Codeigniter*, *CakePHP*, *WebWork*, Spring, *Yii*, dan sebagainya. Setelah dilakukan implementasi arsitektur *web* dengan *load* dan *scalability testing* menggunakan Apache Jmeter, waktu rata-rata yang dibutuhkan *web* Model 1 untuk merespon permintaan *user* lebih cepat dan efisien dengan jumlah *Concurent user (Number of Thread*) yang banyak dalam waktu bersamaan dibanding model 2 (MVC) begitu juga pada penggunaan *resource* memori dan CPU Model 1 lebih unggul. Model 2 memerlukan waktu untuk melakukan *load library* dan parser kode (*source*) dalam hal interaksi pola karena pemisahan bagian kode yaitu *Model – View – Controller*. Selain itu, perubahan nilai *stack size* pada sistem operasi juga mempengaruhi kinerja aplikasi *web* dimana semakin tinggi nilai *stack size* maka kebutuhan memori semakin besar pula.

2. METODE

2.1. *Model View Controller* (MVC)

Untuk membangun aplikasi *web* sebelumnya menggunakan konsep atau arsitektur yang disebut dengan arsitektur Model 1.

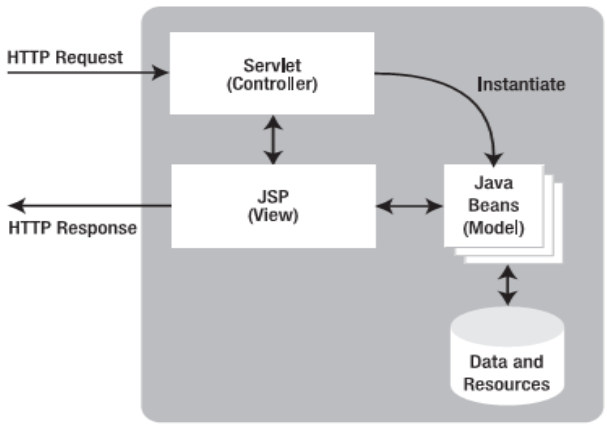


Gambar 1. Arsitektur Model 1

Dalam arsitektur Model 1, *HTTP request* diterima oleh sebuah komponen *web* (misalnya *Java Server Page (JSP)* atau *servlet*). Kemudian *request* diproses dan langsung dihasilkan response yang biasanya berupa halaman HTML. Selain itu dalam arsitektur Model 1, logika bisnis **(***business-logic)* dan presentasi (*view*) digabungkan dalam satu komponen *web*. Hal ini tidak menjadi masalah jika logika bisnis dan presentasi masih relatif sederhana. Namun jika logika menjadi semakin kompleks, pendekatan ini bisa menjadi masalah. Arsitektur Model 2, atau *Model View Controller* (MVC) kemudian diperkenalkan untuk mengatasi masalah tersebut [1]. MVC adalah pola desain atau arsitektur yang digunakan dalam rekayasa perangkat lunak, dimana terjadi pemisahan yang jelas antara data (*model*), dengan *user* interface (*view*) [2]. MVC merupakan model atau metode untuk membuat aplikasi dengan memisahkan data (*model*) dari tampilan (*view*) dan bagaimana memprosesnya (*controller*) [3].

Arsitektur Model 2 sebenarnya implementasi *server-side* dengan membagi aplikasi menjadi bagian-bagian tersendiri yaitu Sumber data sebagai *Model*, representasi atau tampilan dengan *View* dan pemrosesan sebagai *Controller*. Lebih jelasnya adalah sebagai berikut:

1. ***Model***, merupakan implementasi dari logika bisnis dan data bisnis. Model dapat direalisasi dengan memakai sembarang komponen *web*.
2. ***View***, merupakan implementasi dari presentasi, yaitu halaman yang akan dipakai sebagai response untuk dikirimkan kepada *client*. *View* akan menampilkan data bisnis yang telah diolah. Umumnya yang paling cocok dipakai adalah halaman JSP.
3. ***Controller***, merupakan pengontrol aliran *request* (data). Tugasnya adalah menerima *request* yang dikirimkan dari *client*. Data *request* akan diolah atau diteruskan kepada komponen lain yang mengolah data. Pada akhirnya *request* yang diolah akan diserahkan kepada komponen *view*. Umumnya dipakai *servlet* sebagai *controller***.**

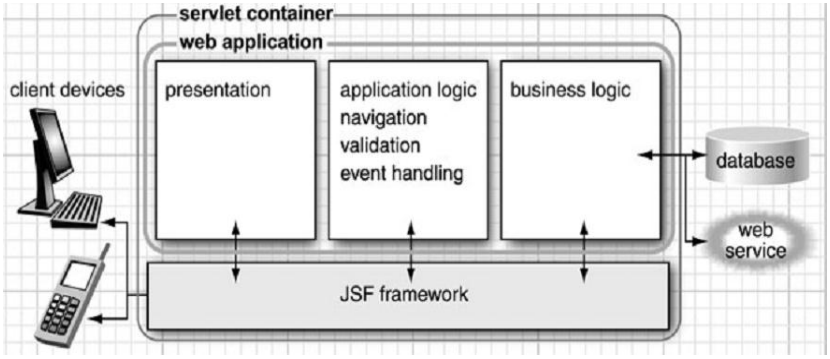


Gambar 2. Arsitektur Model 2 (MVC)

Arsitektur MVC ini memungkinkan adanya perubahan dalam domain model tanpa harus mengubah kode untuk menampilkan domain model tersebut. Hal ini sangat bermanfaat ketika aplikasi mempunyai domain *model* dan *view* komponen sangat besar dan kompleks. Sehingga konsep MVC diperkenalkan untuk teknologi aplikasi *web* khususnya berskala besar atau kompleks (*enterprise*).

2.2. *Java Server Faces*

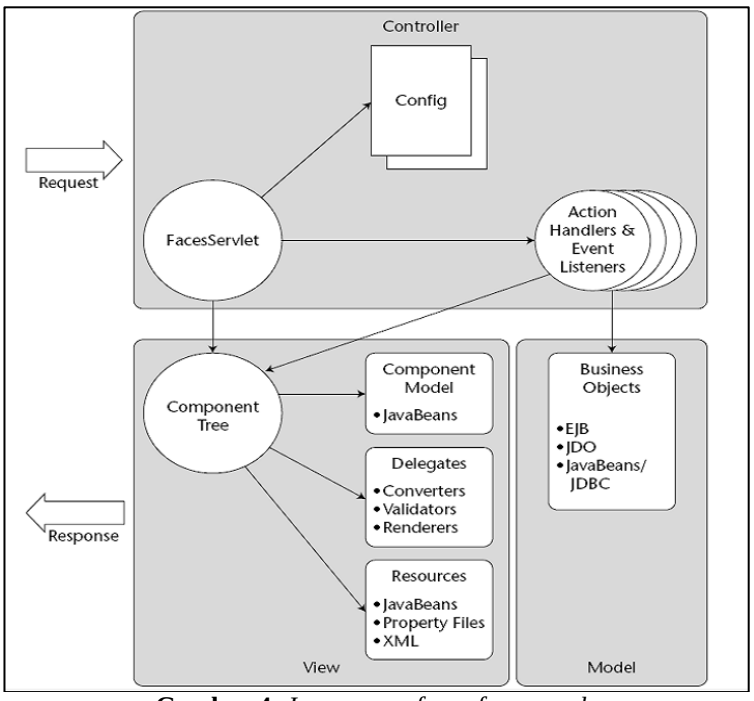
Menurut Bauer [4] *Java Server Faces* (JSF) adalah *user interface framework* dalam bahasa Java untuk membangun aplikasi *web*. JSF merupakan salah satu bagian dari teknologi pada *platform* Java EE. JSF diciptakan pada tahun 2002 melalui *Java Specification Request* (JSR) 127. Salah satu kelebihan utama dari JSF adalah teknologi ini menawarkan pembagian yang jelas antara layer presentasi dan bisnis.



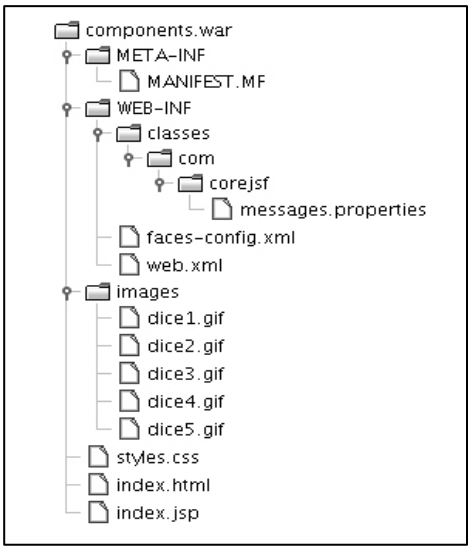
Gambar 3. Arsitektur *Java Server Faces*

Pada Gambar 3 di atas terlihat bahwa JSF bertanggung jawab dalam menangani interaksi klien dan aplikasi, menghubungkan bagian presentasi, *logic* aplikasi dan *bussiness logic* menjadi suatu aplikasi *web*. Sub sistem lainnya, seperti layanan EJB atau basis data dapat diintegrasikan dengan mudah walaupun bukan merupakan bagian dari JSF.

Proses detail *framework* JSF bekerja dapat diilustrasikan pada Gambar 4 dan Gambar 5.

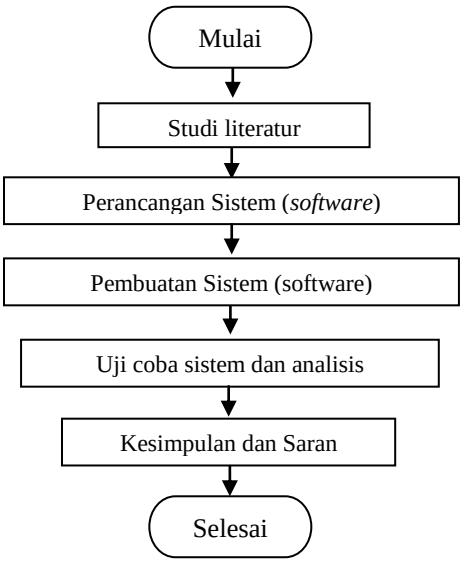


Gambar 4. *Java server faces framework*



Gambar 5. Struktur direktori *Java Server Faces*

Implementasi model 2 atau *model view controller* (MVC) telah dilakukan pada sistem informasi katalog *online* atau *Online Publik Access Catalog* (OPAC) studi kasus Universitas Islam Negeri Alauddin Makassar. Pada tahapan ini lebih menekankan analisis performansi arsitektur aplikasi *web*site dengan metode uji *Load dan Scalability Test* pada Model 2 atau *Model View Controller* (MVC) dan Model 1 sebagai pembanding dengan penerapan metode mengikuti tahapan seperti pada Gambar 6.



Gambar 6. Diagram tahapan

Tahapan secara garis besar dijelaskan sebagai berikut:

a. Studi literatur, tahap ini untuk mencari informasi konsep pengujian sistem pada aplikasi *web*site dengan metode *Load* dan *Scalability Testing* menggunakan Apache Jmeter, maka perlu dilakukan studi literatur dari berbagai sumber.
b. Perancangan sistem, tahap ini dilakukan perancangan sistem aplikasi *web* dengan menggunakan konsep atau arsitektur Model 1 dan Model 2 (MVC), untuk mendukung proses pengujian dan analisis.
c. Pembuatan sistem, tahap ini merupakan proses pembuatan program sistem aplikasi *web* dengan menggunakan konsep atau arsitektur Model 1 dan Model 2 (MVC).
d. Uji coba sistem dan analisis, tahap pengujian dilakukan untuk menguji kerja dari keseluruhan sistem menggunakan metode *Load* dan *Scalability* Testing yang mencakup:
    1. Pengujian terhadap *response time* terhadap peningkatan jumlah *user* yang menggunakan sistem secara bersamaan (*concurrent users*) sekaligus penggunaan memori dan CPU.
    2. Pengujian terhadap *response time* terhadap peningkatan ukuran *database* sekaligus penggunaan memori dan CPU.

Tahap analisis dilakukan untuk menganalisa hasil pengujian sistem terhadap efisiensi dan efektifitas performansi arsitektur aplikasi *web* yang menggunakan arsitektur Model 2 atau *Model View Controller* (MVC) pada *framework Java Server Faces* (JSF).

3. HASIL DAN PEMBAHASAN

3.1. Hasil Pengujian *Load dan Scalability*

Hasil pengujian yang telah dilakukan dengan menggunakan 1 komputer *client* dapat dilihat pada Tabel 1.

Tabel 1. Data hasil pengujian

Web Statis								
Arsitektur	**Samples**	**Average**	**Min**	**Max**	**Std. Dev.**	**Error %**	**Throughput**	**KB/sec**
	1	6	6	6	0	0.00%	166.7	661.62
MODEL 1	100	4	3	11	1.09	0.00%	1	4.01
	500	3	2	13	1.21	0.00%	5	19.85
	1	18	18	18	0	0.00%	55.6	258.14
MODEL 2	100	13	11	22	1.6	0.00%	1	4.69
	500	10	7	164	7.04	0.00%	5	23.25
Web Dinamis 1								
Arsitektur	**Samples**	**Average**	**Min**	**Max**	**Std. Dev.**	**Error %**	**Throughput**	**KB/sec**
	1	270	270	270	0	0.00%	3.7	27.21
MODEL 1	100	14	9	220	20.77	0.00%	1	7.42
	500	12	10	27	1.74	0.00%	5	36.76
	1	189	189	189	0	0.00%	5.3	64.47
MODEL 2	100	108	53	1585	219.65	0.00%	1	12.3
	500	2826	0	18652	5296.33	69.60%	5	24.87
Web Dinamis 2								

Arsitektur	Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec
	1	714	714	714	0	0.00%	1.4	10.29
MODEL 1	100	17	14	46	3.85	0.00%	1	7.42
	500	14	11	80	3.24	0.00%	5	36.77
	1	1578	1578	1578	0	0.00%	0.6	7.66
MODEL 2	100	221	77	3334	502.57	0.00%	1	12.15
	500	232	0	7128	979.32	89.80%	5	14.38

* Size *database* tabel buku 915 KB atau 6015 data.
** Size *database* tabel buku 1,9 MB atau 12027 data.

Selain melakukan pengujian *Load* dan *Scalability Testing* menggunakan Apache Jmeter, dilakukan pula uji performa penggunaan *resource* memori dan CPU dengan memberikan parameter jumlah *thread* 100, 200, 400, 800, 2.000, dan 5.000 dimana parameter *stack size* yang digunakan adalah 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB dan 64 MB. Hal ini dilakukan karena konsep pengujian pada aplikasi menggunakan konsep *multithreading* dimana suatu kemampuan yang memungkinkan beberapa kumpulan instruksi atau proses dapat dijalankan secara bersamaan yang membutuhkan alokasi memori sehingga nilai *stack size* perlu menjadi pertimbangan dalam pengujian ini.

Hasil pengujian untuk setiap Model menggunakan Apache Jmeter dapat dilihat pada Tabel 2 (Model 1) dan Tabel 3 (Model 2).

Tabel 2. Hasil penggunaan *resource memory* dan CPU pada Model 1

Jumlah *Thread*	*Stack size* (MB)	Resource	
		CPU (%)	Memory (MB)
100	1	13.30	322.95
	2	11.35	945.08
	4	11.11	1324.36
	8	11.43	1206.81
	16	11.21	1337.04
	32	11.13	1481.71
	64	11.62	1620.68
200	1	13.98	336.55
	2	12.65	947
	4	11.92	1296.69
	8	11.94	1206.76
	16	11.96	1330.1
	32	12.28	1471.78
	64	12.05	1623.15
400	1	18.39	338.31
	2	14.12	948.72
	4	11.92	1296.69
	8	13.79	1207.7
	16	13.65	1309.51
	32	13.69	1617.73
	64	13.88	1623.87

Jumlah *Thread*	***Stack size* (MB)**	**Resource**	
		CPU (%)	**Memory (MB)**
800	1	22.87	371.16
	2	19.08	900.41
	4	18.59	1070.69
	8	18.61	1178.85
	16	18.28	1245.38
	32	18.33	1624.15
	64	18.11	1625.25
2000	1	31.15	596.88
	2	36.12	1047.44
	4	32.25	1102.09
	8	34.42	1306.11
	16	34.44	1347.87
	32	31.14	1630.1
	64	28.87	1665.83
5000	1	*error*	*error*
	2	*error*	*error*
	4	*error*	*error*
	8	*error*	*error*
	16	*error*	*error*
	32	*error*	*error*
	64	*error*	*error*

Tabel 3. Hasil penggunaan *resource memory* dan CPU pada Model 2

Jumlah *Thread*	***Stack size* (MB)**	**Resource**	
		CPU (%)	**Memory (MB)**
100	1	12.64	365.43
	2	12.82	495.57
	4	11.31	638.26
	8	11.15	937.17
	16	11.45	999.55
	32	11.03	1068.39
	64	11.42	1141.84
200	1	14.72	365.53
	2	13.54	496.76
	4	12.01	641.23
	8	12.33	938.06
	16	12.01	1002.83
	32	11.94	1069.42
	64	11.81	1142.92
400	1	15.34	369.96
	2	14.62	515.58
	4	12.34	657.35
	8	13.54	937.5
	16	13.43	1007.09
	32	13.62	1071.35
	64	13.81	1144.83
800	1	22.24	404.96
	2	13.73	567.38
	4	24.43	858.33

Jumlah *Thread*	***Stack size* (MB)**	**Resource**	
		CPU (%)	**Memory (MB)**
	8	19.63	957.05
	16	17.31	1019.58
	32	17.22	1076.21
	64	16.61	1149.4
2000	1	25.48	584.92
	2	18.93	735.48
	4	18.02	898.36
	8	18.01	1073.45
	16	17.93	1119.58
	32	17.91	1176.21
	64	17.61	1249.4
5000	1	*error*	*error*
	2	*error*	*error*
	4	*error*	*error*
	8	*error*	*error*
	16	*error*	*error*
	32	*error*	*error*
	64	*error*	*error*

Untuk memudahkan dalam proses analisis sistem, akan diambil sampel salah satu modul yang ada pada masing-masing model atau sistem yaitu halaman *Home* untuk model statis dan halaman katalog buku untuk model dinamis untuk mempermudah menganalisa dan membandingkan alur (*flow*) pada model arsitektur 1 dan model arsitektur 2 atau MVC. Proses pengujian dilakukan dengan menggunakan konsep *client server*, dimana pengujian dilakukan pada 1 komputer *server* dan 1 komputer sebagai *client*.

Alur (*flow*) dalam jalannya sistem model 1 dan model 2 dapat ditunjukkan pada Tabel 4.

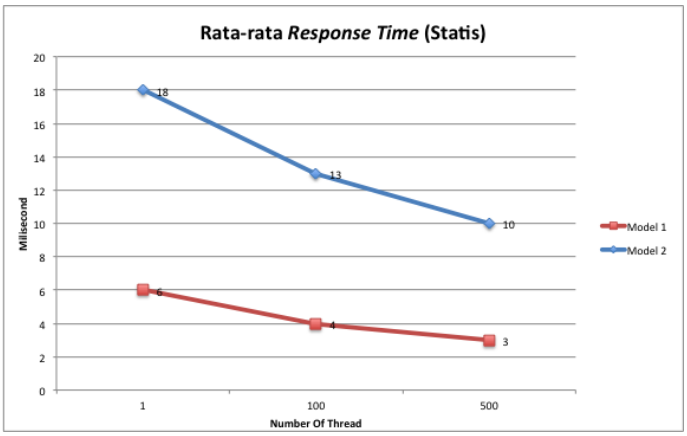
Tabel 4. Perbandingan alur (*flow*) jalannya sistem

Arsitektur Model 1	**Arsitektur Model 2 (MVC)**
Halaman Statis	
Pengunjung/anggota (*user*) – JSP – DAO – JSP – Pengunjung/anggota (*user*)	Pengunjung/anggota (*user*) – JSP – *faces-config.xml* – DAO – JSP – Pengunjung/anggota (*user*)
Halaman Dinamis	
Pengunjung/anggota (*user*) – JSP – DAO – *Database* – DAO – JSP – Pengunjung/anggota (*user*)	Pengunjung/anggota (*user*) – JSP – *faces-config.xml*– DAO – *Database* – DAO – JSP – Pengunjung/anggota (*user*)

Pada pengujian halaman statis pada kedua sistem, model 1 lebih unggul dalam rata-rata waktu merespon permintaan (*response time*) dibandingkan pada model 2 dengan parameter yang sama yaitu *Concurent user (Number of Thread*) 1, 100, 500 *user* dengan permintaan data (*loop count*) 1 dan jumlah periode *ramp-up* masing-masing

10 detik. Pada pengujian ini semua permintaan data dapat direspon dan diproses oleh sistem.

Jika dilihat secara seksama Model 2 terlihat jelas *faces-config.xml* dan *Action* mengambil peranan penting dalam alur (*flow*) jalannya sistem. Pada bagian JSP hanya mengambil data permintaan dari *user* dan menampilkannya kembali pada *user* sehingga membutuhkan waktu tanggap (*response time*) yang lebih lama dibandingkan dengan Model 1 dimana alur jalannya sistem lebih sederhana karena peran JSP sangat bekerja keras menangani permintan (*request*) data sampai menampilkan kembali pada pengunjung atau anggota (*user*). Perbandingan rata-rata pengujian ini dapat dilihat pada Gambar 7.

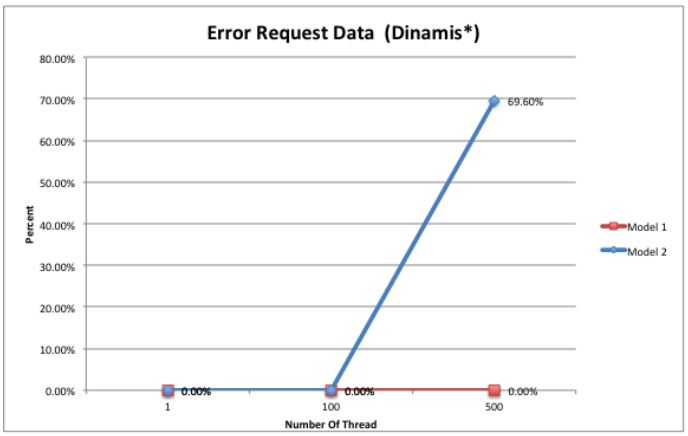


Gambar 7. Grafik waktu rata-rata dari satu kali hasil pengujian pada halaman statis Model 1 dan Model 2

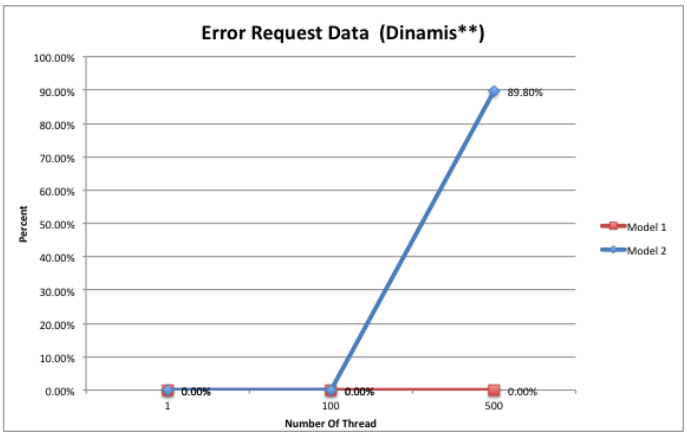
Selain perbedaan alur (flow) sistem pada model 1 dan model 2, arsitektur *web* yang menggunakan model 2 juga membutuhkan waktu dalam proses *load* semua *library* yang ada pada *framework* JSF baik yang digunakan oleh sistem ataupun tidak digunakan, sehingga untuk memberikan waktu tanggap model 1 lebih cepat dan efisien.

Untuk pengujian halaman dinamis pada kedua sistem, model 1 masih tetap lebih unggul dalam waktu rata-rata merespon permintaan (*respon time*) dibandingkan pada model 2 dengan parameter yang sama yaitu *concurent user (Number of Thread*) 1, 100, 500 *user* dengan permintaan data (*loop count*) 1 dan jumlah periode *ramp-up* masing-masing 10 detik. Pada pengujian ini *Concurent user* jumlah 1 dan 100 *user* seluruh permintaan data dapat direspon dan diproses oleh sistem. Sedangkan *Concurent user (Number of Thread*) 500 pada Model 2 mengalami *bottleneck* dimana sistem dan layanan server tidak dapat merespon permintaan *user*, dimana parameter pengujian yang terdiri dari 6.015 data persentasi jumlah permintaan yang gagal 69,60% dengan waktu tunggu (*delay*) 1 detik, demikian pada jumlah data 12.027 persentasi jumlah permintaan yang gagal adalah 89,80% dengan waktu tunggu 0,2 detik. Sebaliknya pada Model 1 dengan *concurent user (Number of Thread*) 500

semua permintaan data berhasil ditampilkan pada sisi pengguna (*user*) dan kegagalan 0%.



Gambar 8. Grafik persentasi jumlah permintaan yang gagal (kesalahan) dengan jumlah 6.015 data



Gambar 9. Grafik persentasi jumlah permintaan yang gagal (kesalahan) dengan jumlah 12.027 data

Penggunaan *Resource Memory* dan CPU

Pada hasil pengujian model 1, jumlah *thread* mempengaruhi penggunaan *resource memory* dan CPU dimana hasil yang didapatkan dengan jumlah *thread* yang sama dengan *stack size* yang sama terjadi peningkatan penggunaan *resource memory* dan CPU. Hasil pengujian ini dapat dilihat pada Tabel 5.

Tabel 5. Eksperimen *Thread* berbeda dengan *Stack size* 1024 KB pada model 1

Jumlah *Thread*	***Stack size* (MB)**	***Resource* CPU**	***Memory* (MB)**	**Ket.**
100	1	13.30	322.95	Berhasil

Jumlah *Thread*	***Stack size* (MB)**	***Resource***		**Ket.**
		CPU	***Memory* (MB)**	
200	1	13.98	336.55	Berhasil
400	1	18.39	338.31	Berhasil
800	1	22.87	371.16	Berhasil
2000	1	31.15	596.88	Berhasil
5000	1	0	0	Berhenti

Begitu pula pada jumlah *thread* yang sama dengan *stack size* yang berbeda terjadi peningkatan penggunaan *resource* memori dimana semakin tinggi nilai *stack size* yang diberikan maka penggunaan *resource* memori juga yang dibutuhkan besar. Akan tetapi *stack size* yang paling banyak penggunaan *resource* CPU terjadi pada *stack size* 1 MB yaitu 13.30%, hal ini terjadi dikarenakan alokasi limit memori yang diberikan sedikit yaitu 1024KB dengan menangani proses *thread* yang banyak sehingga beban pada CPU akan meningkat seperti terlihat pada Tabel 6.

Tabel 6. Eksperimen *Thread* 100 dengan *stack size* berbeda pada Model 1

Jumlah *Thread*	***Stack size* (MB)**	***Resource***		**Ket.**
		CPU	***Memory* (MB)**	
100	1	13.30	322.95	Berhasil
	2	11.35	945.08	Berhasil
	4	11.11	1324.36	Berhasil
	8	11.43	1206.81	Berhasil
	16	11.21	1337.04	Berhasil
	32	11.13	1481.71	Berhasil
	64	11.62	1620.68	Berhasil

Hal serupa juga berlaku pada pengujian arsitektur model 2, dimana penggunaan *resource memory* dan CPU meningkat pada jumlah *thread* dan *stack size* yang nilainya lebih besar .

4. SIMPULAN

Telah dibuktikan bahwa dalam implementasi arsitektur *web* dengan *load* dan *scalability testing* menggunakan Apache Jmeter, waktu rata-rata yang dibutuhkan *web* Model 1 untuk merespon permintaan *user* lebih cepat dan efisien dengan jumlah *concurent user (number of thread*) yang banyak dalam waktu bersamaan dibanding model 2 (MVC) begitu juga pada penggunaan *resource* memori dan CPU Model 1 lebih unggul. Model 2 memerlukan waktu untuk melakukan *load library* dan parser kode (*source*) dalam hal interaksi pola karena pemisahan bagian kode yaitu *Model – View – Controller*, semakin banyak dilakukan pemisahan dan penggunaan library

pada implementasi model *web* maka waktu untuk merespon permintaan *user* mengalami nilai *response time* yang tinggi dan membutuhkan *resource memory* dan CPU yang besar. Selain itu, perubahan nilai *stack size* pada sistem operasi juga mempengaruhi kinerja aplikasi *web* dimana semakin tinggi nilai *stack size* maka kebutuhan memori semakin besar pula

5. REFERENSI

[1] Gunawan. 2010. Implementasi Model *View* Controller (MVC) menggunakan *Framework* JavaServer Faces (JSF) pada *Web* Perpustakaan UIN Alauddin Makassar. *Proceedings: International Conference on Education Technology Strengthening (IC-ETS) 2011*. State University of Malang.

[2] Widiyanto, N. 2010. *Membangun Aplikasi Java Enterprise dengan Arsitektur Model View Controller (MVC).* Yogyakarta: Andi.

[3] Utpatadevi, Sudana, dan Cahyawan. 2012. Implementation of MVC (Model-View-Controller) Architectural to Academic Management Information System with Android Platform Base. *International Journal of Computer Applications*. Vol. 57(8): 1-6.

[4] Bauer, C., King, G. 2007. *Java Persistence with Hibernate*. New York: Manning Publications Co.