PERANCANGAN DAN PENGUJIAN ARSITEKTUR KOMPONEN MULTI-TIER TERDISTRIBUSI UNTUK SMILE SERVER BERBASIS JAVA ENTERPRISE EDITION

SKRIPSI

Oleh

BAHRUN ULUM

0403030225

DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNIK UNIVERSITAS INDONESIA
GANJIL 2007/2008

PERANCANGAN DAN PENGUJIAN ARSITEKTUR KOMPONEN MULTI-TIER TERDISTRIBUSI UNTUK SMILE SERVER BERBASIS JAVA ENTERPRISE EDITION

SKRIPSI

Oleh

BAHRUN ULUM

0403030225

SKRIPSI INI DIAJUKAN UNTUK MELENGKAPI SEBAGIAN PERSYARATAN MENJADI SARJANA TEKNIK

DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNIK UNIVERSITAS INDONESIA
GANJIL 2007/2008

PERNYATAAN KEASLIAN SKRIPSI

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul :

PERANCANGAN DAN PENGUJIAN ARSITEKTUR KOMPONEN MULTI-TIER TERDISTRIBUSI UNTUK SMILE SERVER BERBASIS JAVA ENTERPRISE EDITION

yang dibuat untuk melengkapi sebagian persyaratan sebagai Sarjana Teknik pada Program Studi Teknik Elektro Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia, sejauh yang saya ketahui bukan merupakan tiruan atau duplikasi dari skripsi yang sudah dipublikasikan atau pernah dipakai untuk mendapatkan gelar kesarjanaan di lingkungan Universitas Indonesia maupun di Perguruan Tinggi atau Instansi manapun, kecuali bagian yang merupakan sumber informasinya dicantumkan sebagaimana mestinya .

Depok, 14 Desember 2007

Bahrun Ulum

NPM 04 03 03 022 5

PENGESAHAN

Skripsi dengan judul:

PERANCANGAN DAN PENGUJIAN ARSITEKTUR KOMPONEN MULTI-TIER TERDISTRIBUSI UNTUK SMILE SERVER BERBASIS JAVA ENTERPRISE EDITION

dibuat untuk melengkapi sebagian persyaratan menjadi Sarjana Teknik pada Program Studi Teknik Elektro Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia. Skripsi ini telah diujikan pada sidang skripsi pada Tanggal 4 Januari 2008 dan dinyatakan memenuhi syarat/sah sebagai skripsi pada Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia.

Depok, 4 Januari 2008

Dosen Pembimbing

Dr.Ing.Kalamullah Ramli, M.Eng

NIP 132 092 429

UCAPAN TERIMA KASIH

Puji syukur kehadirat Allah SWT atas karunia dan rahmat-Nya skripsi ini dapat penulis selesaikan dengan baik. Sholawat dan salam penulis sampaikan kepada Rosulullah Muhammad SAW, sebagai pembawa rahmat dan menjadi suri tauladan yang terbaik bagi selurh alam. Penulis juga mengucapkan terima kasih kepada:

Dr.Ing.Kalamullah Ramli, M.Eng

selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberikan saran, pengarahan, dan bimbingan serta persetujuan sehingga skripsi ini dapat selesai dengan baik.

Bahrun Ulum
NPM 04 03 03 0225
Departemen Teknik Elektro
Universitas Indonesia

Dosen Pembimbing
Dr.Ing.Kalamullah Ramli, M.Eng
NIP 132 092 429

PERANCANGAN DAN PENGUJIAN ARSITEKTUR KOMPONEN MULTI-TIER TERDISTRIBUSI UNTUK SMILE SERVER BERBASIS JAVA ENTERPRISE EDITION

ABSTRAK

Perkembangan *mobile* teknologi yang semakin terintegrasi berdampak besar pada proses belajar-mengajar. Keadaan ini memungkinkan ekstensifikasi proses belajar-mengajar dari dalam ruangan ke keluar ruangan kelas, baik secara *real* maupun *virtual* sesuai dengan situasi dan keinginan pribadi pelakunya serta dapat dilakukan setiap saat. Adanya lingkungan yang mampu mendukung situasi tersebut dengan menawarkan layanan-layanan yang dapat mendukung sistem *mobile learning* mutlak diperlukan.

Pada skripsi ini dibangun sebuah sistem terintegrasi yang memberikan layanan-layanan kepada *end-user* sehingga konsep *mobile learning* sangat mengkin untuk diterapkan dikehidupan sehari-hari. Sistem ini dinamakan SMiLE. SMiLE dibangun berdasarkan arsitektur komponen *multi-tier* yang berinterkasi satu sama lain secara independen. *MIDlet* dan klien *web service* berjalan pada sisi klien; *servlet, JSP, Web service* dan *EJB* berjalan pada sisi *server* serta *Derby database* sebagai sistem *back-end*.

Hasil uji coba terhadap SMiLE, menunjukkan kemampuan SMiLE dalam melakukan proses *multithread* hingga *thread* yang aktif pada waktu yang bersamaan mencapai 54 *thread*. Adanya *garbage collector*, membuat penggunaan memori pada SMiLE menjadi lebih efisien, hanya menggunakan 45,9 *Megabyte* dari 54,9 *Megabyte* memori yang disediakan sistem sehingga tak pernah terjadi keurangan memori (*leak*).

Kata kunci : SMiLE, Java ME, Java EE, *Web Services*, CLDC, MIDP, EJB, *Servlet*, JSP, *Mobile learning*, GPRS.

Bahrun Ulum
NPM 04 03 03 0225
Department of Electrical Engineering
University of Indonesia

Counsellor
Dr.Ing.Kalamullah Ramli, M.Eng
NIP 132 092 429

DEVELOPING AND TESTING DISTRIBUTABLE MULTI-TIER COMPONENT ARCHITECTURE FOR SMILE SERVER BASED ON JAVA ENTERPRISE EDITION

ABSTRACT

The development of mobile tecnology that more integrated has big effected to learning process. This situation allow exstensification of learning process from inside to outside classroom, whether by real or virtual depend on the situation and individual interest, also can be done anytime. The environment that could support the situation by offering services which could support mobile learning system was absolutely needed.

In this final project is built an integrated system which could give services to end user, hence the mobile learning concept is very possible to be implemented in daily life. The system is named SMiLE. SMiLE is built based on multi-tier component architecture that could interact from one to the other independently. MIDlet and web service client operated on client side; servlet JSP, Web service and EJB operated on server side also Derby database as back-end system.

The experiment results showed that SMiLE ability in doing multithread process until the active thread in the same time reach 54 threads. By having garbage collector, make the memory usage on SMiLE become more efficient, that is only use 45.8 Megabyte from 54.9 Megabyte memory that is available on system, hence the system never lack of memory (leak of memory).

Keywords : SMiLE, Java ME, Java EE, Web Services, CLDC, MIDP, EJB, Servlet, JSP, Mobile learning, GPRS.

DAFTAR ISI

Halaman

DAFTAR GAMBAR

DAFTAR TABEL

Halaman

DAFTAR SINGKATAN

API	Application Programming Interface
EJB	Enterprise JavaBean
EIS	Enterprise Information System
ERP	EnterpriseResource Planning
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
GPRS	Global Packet Radio System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HTML	Hyper text Mark-up Language
HTTP	Hypertext Transport Protocol
IDL	Interface Definition Language
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
Java EE	Java Enterprise Edition
Java ME	Java Micro Edition
JDK	Java Developer Kit
JNDI	Java Naming and Directory Interface
JSP	JavaServer Page
JVM	Java Virtual Machine
MMS	Multimedia Messaging Service
MS	Mobile station
PDA	Personal Digital Assistant
RAM	Random Access Memory
RMI	Remote Method Invocation
SMS	Short Messaging Service

SMiLE	System of Mobile Learning Environment
SOAP	Simple Object Uccess Protocol
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
WAP	Wireless Application Protocol
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WML	Wireless Markup Language
WSDL	Web Service Description Language
xHTML	Extensible HyperText Markup Language
XML	eXtensible Mark-up Language

DAFTAR ISTILAH

Application Server	Komponen software yang digunakan untuk menjalankan aplikasi pada sisi server seperti servlet, JSP dan EJB.
Applet	Komponen aplikasi klien-server yang berjalan pada sisi server.
Bandwith	Lebar kanal atau pita
Bluetooth	Bluetooth adalah sebuah teknologi komunikasi wireless (tanpa kabel) yang beroperasi dalam pita frekuensi 2,4 GHz unlicensed ISM dengan jangkauan area 10 - 100 m.
Bytecode	Hasil kompilasi program kode Java (*file.java*)
Cluster	Bagian dari Distributed Programming
Computer Network	Jaringan komputer.
Compatible	Sesuai, cocok.
Concurrent Request	Request yang terjadi secara bersamaan.
Database	Suatu metoda yang dipilih untuk penyimpanan aplikasi besar yang digunakan bersama-sama, multiuser dimana dibutuhkan koordinasi antar banyak user.
Deployment-Descriptor	Komponen software yang berisi informasi tentang komponen software lainnya.
Digital	Kondisi nol dan satu.
Distance-learning	Proses belajar mengajar dari jarak jauh
Distributed-Programming	Suatu sistem komputasi yang terdiri dari dua komputer atau lebih yang digunakan untuk menyelesaikan suatu permasalahan komputasi yang besar.
E-learning	Istilah lain dari “electronic learning”, yaitu teknik pembelajaran dengan memanfaatkan dunia maya sebagai jembatan antara pengajar dan yang diajar.
Grid Computing	Bagian dari Distributed Programming (Computing)

Handphone	Pesawat telepon yang mudah dipindah-pindah
Interpreted	Eksekusi bytecode
JavaBeans	Komponen software yang dapat digunakan kembali dan biasanya terdiri dari method set dan get untuk mengaksesnya.
Laptop	Komputer jinjing
Method	Prosedur, Komponen Kode program yang membuat program lebih terstruktur.
Microsoft Pocket PC	Sistem operasi pada PDA
Mobile	Mudah dipindah-pindah atau dibawa-bawa
Mobile Device	Devais berupa notebook, PDA, hanphone, smartphone dan mudah dipindah-pindah
Mobile learning	Proses belajar mengajar yang menggunakan perangkat mobile sebagai medianya
Multimedia	Suatu aplikasi berupa gambar dan atau video dan atau suara.
Multi-tier	Arsitektur komponen bertingkat
Multitasking	Melakukan beberapa kerja dalam satu waktu.
Multithread	Kemampuan menjalankan method dalam satu waktu tertentu.
Notebook	Komputer jinjing.
Oak	Nama awal Java.
Object-oriented	Pemrograman bebasis objek
Packet-switched	Suatu teknologi yang mengirimkan data dalam bentuk paket-paket
Palm OS	Sistem operasi pada PDA
Processor	Unit pengolah data
Programmer	Orang yang membuat program, pemrogram
Real-time	Komunikasi dua klien dalam waktu yang bersamaan
Reliable	Tahan terhadap berbagai gangguan hingga nilai tertentu
Servlet	Kelas dalam Java yang digunakan untuk memperluas kemampuan server agar mudah diakses melalui model

	pemrograman request-response dan berjalan pada sisi server.
Session	Waktu ketika aplikasi dijalankan
Sistem Operasi	Sistem operasi adalah sebuah sistem yang diperlukan untuk dapat menjalankan semua aplikasi program atau software yang ada di komputer atau embeded system.
Smartphone	Handphone yang dilengkapi dengan sistem operasi
Stand Alone	Program yang berjalan tanpa hubungan dengan program lain
Symbian OS	Sistem operasi pada smartphone/PDA phone
Tablet PC	Personal Computer yang dibentuk ringkas seperti laptop.
Thin Client	Aplikasi klien ringan seperti HTML dan xHTML
Web Browser	Aplikasi yang digunakan untuk menjalankan halaman web.
Web Service	Aplikasi berskala besar (enterprise application) berbasis web yang menawarkan koleksi layanan yang dapat diakses oleh klien melalui pesan berbasis XML
Wi-Fi	Standar transfer data dalam area lokal
Wi-Max	Standar transfer data dalam area lebih luas dari lokal
2G	Standar komunikasi mobile generasi kedua
3G	Standar komunikasi mobile generasi ketiga

BAB I
PENDAHULUAN

1.1 LATAR BELAKANG MASALAH

Dunia telah berada dalam genggaman. *Mobile* divais yang biasa berada dalam saku, mampu terhubung jaringan internet yang hampir selalu ada dimanapun kita berada. Begitu juga dengan aplikasi informasi dan teknologi komunikasi pada perseroan maupun sektor publik kini telah menjadi wacana besar, meskipun industri ini masih relatif baru. Perkembangan teknologi baru ini bahkan lebih cepat daripada kemampuan masyarakat untuk menerapkan infrastruktur teknologi tersebut. Kini penetrasi dan adopsi *mobile* divais pada setiap segmen kehidupan terjadi: *PDA (personal Digital Assistant), smartphone, handphone, MP3 player, portable game device, tablet PC* dan *laptop.* Hal ini terlihat dari penambahan jumlah pengguna *mobile* divais yang diprediksi lebih pesat dari pengguna internet. Menurut sumber dari ITU, pada Tahun 2005, pelanggan *mobile* divais dari penduduk dunia adalah 33 %, sedangkan perkiraan Tahun 2010 adalah 47 %. Sementara itu, pelanggan internet Tahun 2005, hanya 13 % dan diperkirakan mengalami perkembangan yang lambat. Pertumbuhan selular terpesat selama lima tahun terakhir akan terjadi di negara India dengan pertumbuhan rata-rata per tahun 80%. Sedangkan Vietnam 62%, Pakistan 38.5%, China 22% dan Indonesia 19.5%.

Gambar 1.1 Metode baru proses belajar mengajar: *mobile learning* [1]

Pesatnya perkembangan teknologi *mobile* ini tentu berdampak juga pada bidang pendidikan. Jika pendidikan tradisional menggunakan proses belajar mengajar di dalam kelas, dimana seorang guru mempresentasikan materi pelajaran kepada para siswa. Sementara *e-learning* adalah proses belajar mengajar yang menggunakan media internet untuk transfer materi pelajaran dan proses tersebut masih berada dalam ruangan khusus. Maka *mobile learning (m-learning)* adalah proses belajar-mengajar yang menggunakan *mobile* divais sebagai media transfer materi pelajarannya, seperti terlihat seperti Gambar 1.1

Mobile berarti sesuatu yang pribadi dan dapat dibawa kemana-mana. Sedangkan *mobile learning* berarti proses belajar-mengajar yang dapat dilakukan secara *mobile* diberbagai tempat; tempat kerja, rumah atau ketika berwisata. *Mobile* terhadap lingkungan sekitar; berhubungan dunia kerja, meningkatkan pengetahuan atau tentang kejadian ditempat wisata dan *mobile* terhadap waktu; sepanjang hari kerja atau akhir pekan.

1.2 TUJUAN

Perkembangan *mobile* teknologi yang semakin terintegrasi, familiar, saling terhubung dan kemampuan meningkatkan interaksi sosial serta konektivitas terhadap internet, dapat berdampak besar pada proses belajar-mengajar. Keadaan ini akan memaksa ekstensifikasi proses belajar-mengajar dari dalam ruangan ke keluar ruangan kelas, baik secara *real* maupun *virtual* sesuai dengan situasi dan keinginan pribadi pelakunya serta dapat dilakukan setiap saat. Oleh karena itulah skripsi ini bertujuan untuk membuat model lingkungan yang mampu mendukung situasi tersebut dengan menawarkan layanan-layanan yang dapat digunakan dalam sistem *mobile learning* serta mampu menangani pertumbuhan klien yang sangat pesat seperti pesatnya pertumbuhan *mobile device*.

1.3 PEMBATASAN MASALAH

Implementasi *m-learning* dengan komponen terdistribusi sangat sulit dilakukan. Hal ini karena pemrogram perlu mengatur bagaimana masing-masing komponen dapat saling berkomunikasi. Misalnya jika dua klien akan mengakses data melalui *enterprise bean* yang sama siapa yang akan didahulukan. Belum lagi

pengaturan terhadap *bean* yang harus menjaga informasi kliennya hingga pada waktu tertentu. Tentu ini akan jauh lebih rumit lagi jika klien yang mengakses sangat banyak, sehingga membutuhkan pengaturan memori yang baik dan pengetahuan akan kemampuan *concurrent request* dari kontainer serta tentunya pengaturan masalah *transaction*. Sehingga penulis tidak akan membahas permasalahan rumit tersebut dan hanya akan membuat model komponen terdistribusi untuk *server* SMiLE agar mampu menangani peningkatan jumlah klien dan bagaimana komponen berkomunikasi.

1.4 SISTEMATIKA PENULISAN

Secara umum skripsi ini terdiri dari dua bagian besar, yaitu teori pengantar, perancangan arsitektur, implementasi dan uji coba. Secara lebih detail, skripsi ini dijabarkan sebagai berikut:

BAB I PENDAHULUAN

Bab ini berisi latar belakang penulisan makalah yang berisi perkembangan teknologi *mobile* dan harapan teknologi *mobile* pada kegiatan belajar mengajar. Selain itu, pembatasan masalah dan sistematika penulisan makalah ini dijelaskan di dalam bab ini.

BAB II LANDASAN TEORI

Bab ini merupakan bagian yang berisi teori pendukung dan penjabaran tentang lingkungan *mobile learning*. Seperti: Java EE, *Web Service*, *Mobile Terminal*, GPRS.

BAB III DESAIN DAN IMLEMENTASI SMILE

Bab ini membahas permasalahan yang ada pada sistem komponen terdistribusi, solusi penyelesaian, membuat arsitekturnya dan bagaimana mengimplementasikan solusi tentang permasalahan yang akan di hadapi dalam membangun jaringan *mobile learning* berdasarkan teori pendukung bab sebelumnya.

BAB IV UJI COBA DAN ANALISA *SERVER* SMILE

Uji coba SMiLE (*System of Mobile Learning Environment*) dengan menggunakan *Netbeans Profiler* (*www.netbeans.org*) dan *Apache JMeter* (*Apache Software Foundation*) serta penjelasan dan analisa terhadap

hasil penjelasan yang di peroleh.

BAB V KESIMPULAN

Bab ini berisi kesimpulan tentang skripsi ini.

BAB II
LANDASAN TEORI

2.1 *MOBILE LEARNING*

2.1.1 Definisi *Mobile Learning*

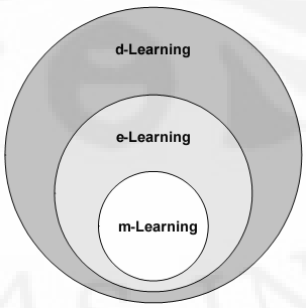
Mobile learning, sampai saat ini, belum mempunyai definisi yang baku. Setiap komunitas *mobile learning* membuat definisi berdasarkan pada pengalaman, penggunaan dan latar belakang mereka. Namun demikian, penulis akan mengutip dari beberapa sumber sebagai berikut :

Clark Quinn :

> *"… The intersection of mobile computing and e-learning: accessible resources wherever you are, strong search capabilities, rich interaction, powerful support for effective learning, and performance-based assessment. E-Learning independent of location in time or space."*

www.wikipedia.org :

> *"Mobile learning is the follow up of e-learning which for its part originates from d-learning (distance education). M-learning is the delivery of learning to students who are not keeping a fixed location or through the use of mobile or portable technology."*



Gambar 2.1 *M-Learning* sebagai subset dari *d-learning*.

Dari kedua definisi diatas, *mobile learning* adalah proses belajar mengajar yang merupakan perkembangan lebih lanjut dari *d-learning* dan *e-learning* dengan

mobile device sebagai antar mukanya. Hal ini dapat dilihat pada Gambar 2.1. *Mobile device* seperti *notebook*, PDA, *smart phone* maupun *handphone*

2.1.2 Konsep *Mobile Learning*

Konsep *M-learning* pada jenjang pendidikan dasar, menengah dan tinggi tentu saja memiliki karakteristik masing-masing dan tidak dapat disatukan sama lain. Akan tetapi terdapat dua konsep yang secara umum dapat diterapkan dari ketiga jenjang tersebut, yaitu :

1. Konsep *mobile learning* difokuskan untuk menyediakan kelas pembelajaran maya yang memungkinkan interaksi antara guru dan siswa ataupun antara dosen dan mahasiswa. Interaksi meliputi penyediaan materi ajar, ruang diskusi, penyampaian tugas dan pengumuman penilaian.
2. Teknologi yang diadopsi sebaiknya efektif sesuai dengan metode pengajaran dan dinilai sebagai sebuah pembaharuan. Selain itu teknologi yang dipilih sebaiknya mudah diakes dan tersedia dengan distrubusi yang merata di lingkungan siswa maupun pengajarnya.

2.2 *JAVA ENTERPRISE EDITION* (*JAVA EE*)

Java adalah bahasa pemrograman yang dikeluarkan oleh *Sun Microsystems, Inc.,* yang merupakan perbaikan dan pengembangan dari bahasa sebelumnya, diantaranya adalah penggabungan fitur memori manajemen dan *garbage collection* dari *smalltalk* dan sintak dari bahasa pemrograman C/C++. Java dikembangkan sejak Agustus 1991 dengan nama awal *Oak*. Kemudian, karena dianggap kurang komersial, Januari 1995 *Oak* diganti menjadi Java [2].

Pada tahun 1998, *Sun Microsystem, Inc.* memperbaiki spesifikasi Java dan memperkenalkan *Java 2 Standar Edition (J2SE)*. J2SE mengalami perbaikan pada fungsi pustaka dan menyediakan dukungan penuh terhadap pembuatan GUI, jaringan, akses *database* dan lainnya. Pada tahun yang sama, *Sun Microsystems, Inc.,* juga juga meluncurkan *Java 2 Enterprise Edition (J2EE).* Jika pada J2SE hanya mampu membangun aplikasi *stand alone*, maka pada J2EE ini *programmer* dapat membangun dan mendistribusikan aplikasi berskala berskala besar, aplikasi jaringan terdistribusi dan aplikasi berbasis *web*. *Sun Microsystems,*

Inc., juga mengeluarkan *Java 2 Micro Edition (J2ME)* yang dipergunakan membuat program untuk perangkat *portable* dengan keterbatasan memori.

2.2.1 Keunggulan Java

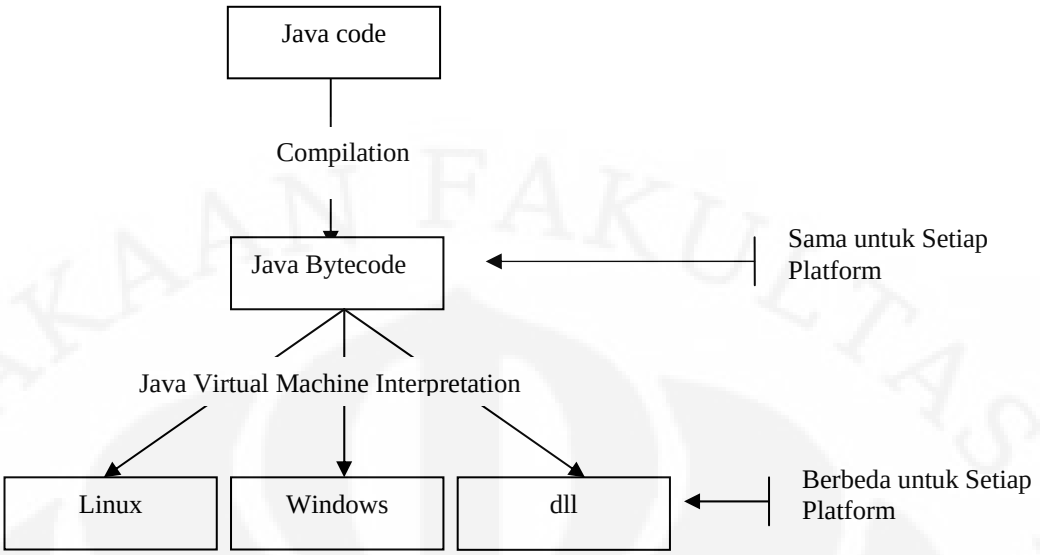
Jika dibandingkan dengan bahasa pemrograman yang telah ada, Java mempunyai beberapa keunggulan seperti berikut ini:

Pemrograman Jaringan. Java menyediakan paket *java.net* yang mempunyai fitur penyedia pelayanan jaringan dan distribusi objek dalam suatu sistem. Melalui paket tersebut, Java menawarkan *stream-based communication* dan *packet-based communication.*

Interpreted. Pada bahasa pemrograman lain, setiap program yang dibuat langsung dieksekusi. Sedangkan pada Java, program yang dibuat harus dikompilasi terlebih dahulu menjadi *bytecode, bytecode* inilah yang kemudian dijalankan oleh *Java Virtual Machine (JVM).* Penggunaan *bytecode* ini membuat Java tidak bergantung pada *platform* tertentu. Hal ini ditunjukkan pada Gambar 2.2.

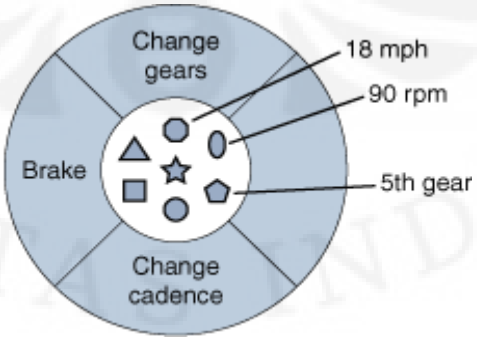
Distributed Programming. Fitur-fitur Java, seperti *Enterprise JavaBean (EJB), Remote Method Invocation (RMI)* dan ketersediaan *IDL-mapping,* sangat potensial untuk membangun aplikasi terdistribusi seperti *cluster* dan *grid computing*.

Multithreading. *Thread* berarti eksekusi prosedur dalam sebuah program. *Multithreading* adalah kemampuan untuk melakukan eksekusi beberapa prosedur atau *method* dalam satu waktu tertentu. *Thread* dalam Java memiliki kemampuan untuk memanfaatkan *multiprocessor*.



Gambar 2.2 Proses pengembanagan program dalam Java

Pemrograman Berorientasi Objek. Pemrograman berorientasi objek menghadirkan dua istilah penting, yaitu kelas dan objek. Pada dunia nyata, objek mempunyai dua karakteristik, yaitu ciri atau keadaan (*state*) dan kebiasaan (*behavior*) [4]. Misalnya sepeda motor mempunyai ciri/keadaan warna, kecepatan, putaran roda dan posisi transmisi. Sedangkan kebiasaannya mengerem, pindah transmisi dan akselerasi. Pada pemrograman berorientasi objek, keadan diterjemahkan sebagai variabel dan kebiasaan diterjemahkan sebagai *method*, yaitu sebuah fungsi yang berhubungan dengan objek tertentu. Misalnya jika ingin menurunkan kecepatan dari 90 km/jam menjadi 60 km/jam, maka kita menggunakan sepeda motor yang mempunyai kebiasaan pengereman (objek sepeda motor, *method* pengereman dan variabel diberi nilai 60 km/jam). Hal ini dapat dijelaskan seperti pada Gambar 2.3.



Gambar 2.3 Model sepeda motor sebagai objek perangkat lunak**.**

Pendekatan pemrograman berorientasi objek ini akan mempermudah pembuatan program, mengurangi duplikasi kode dan dapat mengurangi kesalahan penulisan.

2.3 KOMPONEN JAVA EE

Aplikasi Java EE terdiri atas beberapa komponen. Komponen Java EE adalah suatu unit perangkat lunak fungsional yang mampu berdiri sendiri, yang dirakit ke dalam suatu kontainer Java EE. Komponen-komponen itu dapat saling berkomunikasi dengan komponen lainnya [5].

Pada bagian ini akan dibahas beberapa komponen Java EE yang akan digunakan dalam mengimplementasikan SMiLE, diantaranya :

2.3.1 Komponen Klien

Komponen klien yang digunakan adalah *web* klien. *Web* Klien atau biasa di sebut *thin-client* (aplikasi klien yang ringan, sederhana), terdiri dari dua bagian yaitu berbagai *markup languange* seperti HTML dan XML serta *web browser* yang mengeksekusi halaman *web* dari *server*, seperti ditunjukkan oleh Gambar 2.4. Selain itu, sebagai pilihan, dapat juga ditambah *JavaBean*^{TM} yang digunakan untuk mengatur aliran data diantara Aplikasi klien atau *applet* dan komponen-komponen yang berjalan pada *server* Java EE, atau diantara komponen *server* dan *database*.

Thin-client pada umumnya tidak dapat mengakses *database*, melakukan aturan bisnis yang rumit, atau berhubungan dengan aplikasi pewarisan (*inheritance*). Ketika *thin-client* digunakan, operasi-operasi berat hanya dikerjakan pada bagian *server*, sehingga hal ini dapat meningkatkan keamanan, kecepatan, pelayanan dan ketahanan dari teknologi *server* Java EE.

Komponen *JavaBean* dapat berupa *server ti*er dan *klien tier*, yang mana tidak diperhitungkan oleh spesifikasi komponen Java EE. Komponen *JavaBean* mempunyai fitur yang dapat diakses dengan *method set* dan *get*. Komponen *JavaBean* yang digunakan biasanya sederhana dalam desain dan penerapan tetapi harus cocok dengan konvensi penamaan dan desain dalam pembentukan arsitektur komponen *JavaBean*.

2.3.1.1 Komponen Web

Komponen *web* Java EE dapat berupa *servlet* atau halaman-halaman yang dibuat oleh teknologi *Java Server Pages* (*JSP*) dan atau *JavaServer Faces* (*JSF*), seperti ditunjukkan oleh Gambar 2.4. *Servlet* adalah kelas dari bahasa pemrograman Java yang menghasilkan *request* dan *response* dinamis. Halaman JSP adalah dokumen yang dibuat dalam teks, dieksekusi sebagai *servlet* tetapi dapat menghasilkan suatu kondisi yang statis. Sedangkan *JavaServer Faces* dibangun pada *servlet* dan teknologi JSP yang menyediakan komponen antar muka pengguna untuk aplikasi *web*.

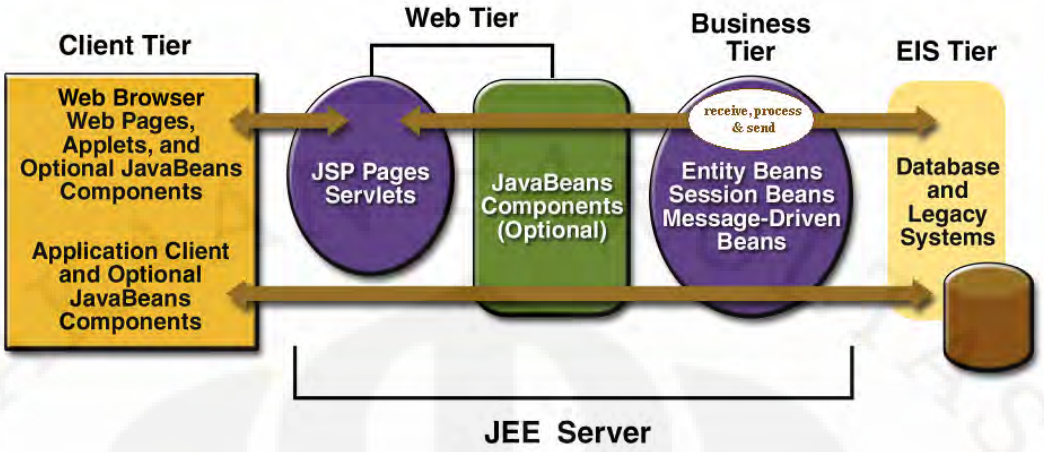
2.3.1.2 Komponen Bisnis

Komponen bisnis berisi aplikasi logika yang berjalan pada sisi *server*, menghubungkan komponen *web* dan *Enterprise Information System (EIS)*. Komponen bisnis dalam skripsi ini adalah *Enterprise JavaBean (EJB)*, terdiri dari : *session bean*, *entity bean*, dan *message-driven bean*.

Komponen bisnis dapat dilihat pada Gambar 2.4, menunjukkan *enterprise bean* menerima, memproses dan mengirimkan data dari program klien *Enterprise Information System (EIS)* untuk disimpan. *Enterprise bean* juga dapat mengambil, memproses dan mengirimkan data dari EIS menuju klien.

2.3.1.3 Komponen Enterprise Information System (*EIS*)

EIS tier menangani perangkat lunak EIS dan sistem infrastruktur *enterprise* yang terdiri dari *Enterprise Resource Planning (ERP)*, proses transaksi *mainframe*, sistem *database*, dan sistem informasi lain. Contohnya ketika aplikasi Java EE memerlukan akses EIS untuk mengambil atau menyimpan *database*.



Gambar 2.4 Interaksi antar komponen Java [6]

Gambar 2.4 diatas menunjukkan elemen-elemen dari klien *tier* yang dapat berkomunikasi dengan *business tier* yang berjalan pada *server* Java EE, baik secara langsung maupun seperti pada *applet* yang menggunakan halaman JSP atau *servlet* yang berjalan pada *web tier*. Penggunaan aplikasi Java EE yang saling berkomunikasi didasarkan kebutuhan fungsi yang diperlukan oleh klien sehingga akan memudahkan dalam melakukan distribusi, pengembangan dan pengaturan sistem.

2.3.2 Kontainer Java EE

Aplikasi *multi-tier* sangat susah untuk ditulis karena melibatkan banyak kode dalam menangani transaksi dan pengelolaan, *multithread*, dan kode-kode spesifik lainnya. Namun, arsitektur Java EE yang terdiri atas komponen-komponen dan *platform* yang bediri sendiri, membuat aplikasi Java EE mudah untuk ditulis. Hal ini disebabkan karena kode-kode Java diatur kedalam komponen yang dapat digunakan ulang. Selain itu, *server* Java EE juga menyediakan pelayanan kontainer, yaitu antar muka antara komponen-komponen serta layanan-layanannya. Sehingga pemrogram tidak perlu lagi membuat layanan-layanan ini, dan hanya konsentrasi pada penyelesaian masalah kode program (*business logic*).

2.3.2.1 Pelayanan Kontainer

Sebelum dieksekusi, sebuah *web, enterprise bean* dan komponen klien harus dirakit ke dalam aplikasi Java EE dan disebarkan kedalam kontainernya. Proses

perakitan melibatkan pengaturan spesifikasi kontainer untuk tiap komponen dalam aplikasi Java EE dan untuk Java EE itu sendiri. Pengaturan kontainer akan mengatur secara manual dukungan yang diberikan oleh *server* Java EE, termasuk masalah keamanan, pengaturan transaksi, pencarian *Java Naming and Directory Interface* (*JNDI*) dan koneksi tersembunyi (*remote conectivity*), yang dapat dijelaskan sebagai berikut:

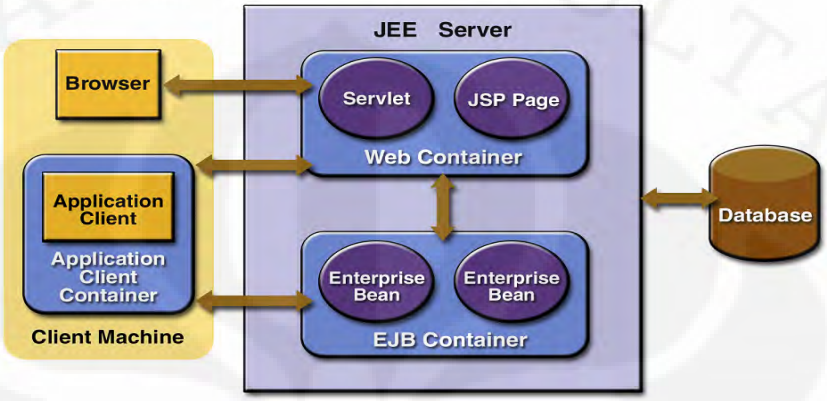
1. Model sistem keamanan Java EE mengizinkan pemrogram untuk mengkonfigurasi komponen *web* atau *enterprise bean* sehingga sistem hanya dapat diakses oleh pengguna yang berhak.
2. Model transaksi Java EE mengizinkan pemrogram untuk menentukan hubungan antar *method*, sehingga semua *method* dalam sebuah transaksi dibentuk kedalam sebuah paket tertentu.
3. Pelayanan JNDI menyediakan sebuah antarmuka yang tetap untuk pelayanan penamaan dan direktori sebuah aplikasi *enterprise* sehingga komponen aplikasi dapat mengakses layanan ini.
4. Model koneksi tersembunyi Java EE mengatur komunikasi tingkat rendah antara klien dan *enterprise bean*. Setelah *enterprise bean* dibuat, sebuah klien memanggil *method* pada *enterprise bean*, seolah-olah berada pada mesin *virtual* yang sama.

Kemampuan pelayanan-pelayanan Java EE yang dapat dikonfigurasi membuat komponen-komponen dalam aplikasi Java EE yang sama dapat bertindak berbeda bergantung dimana mereka diletakkan. Contohnya, sebuah *enterprise bean* mempunyai pengaturan keamanan yang dapat mengizinkan seseorang mengakses *database* pada tingkat tertentu dan seseorang lain pada tingkatan berbeda. Kontainer juga mengatur layanan-layanan yang tidak bisa dikonfigurasi seperti daur hidup *enterprise bean* dan *servlet*, sumber koneksi *database*, dan akses ke Java EE API

2.3.2.2 Tipe-tipe Kontainer

Penyebaran aplikasi komponen Java EE ke kontainer Java EE terlihat seperti Gambar 2.5.

1. ***Server* Java EE**. Merupakan bagian dari produk Java EE yang menyediakan *Enterprise JavaBean (EJB)* dan kontainer-kontainer *web*.
2. **Kontainer *EJB***. Mengatur eksekusi *enterprise bean* untuk aplikasi Java EE. *Enterprise bean* dan kontainernya berjalan pada *server* Java EE.



Gambar 2.5 Java EE *Server* dan Kontainer [7]

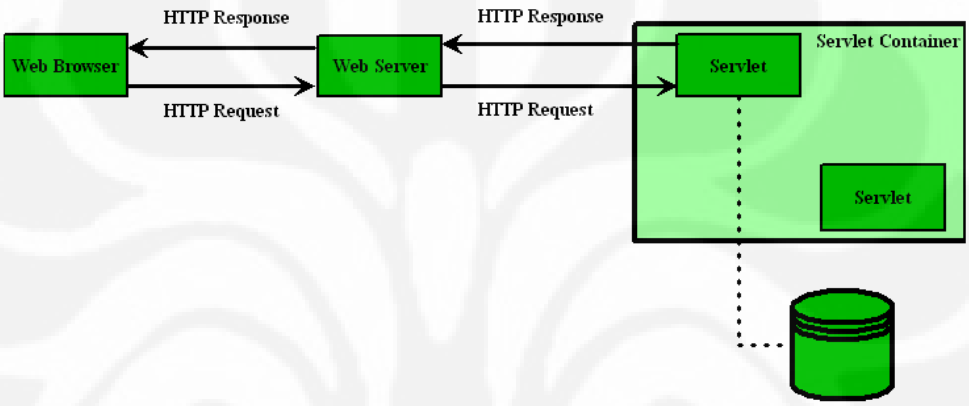
3. **Kontainer *Web*.** Mengatur eksekusi komponen halaman JSP dan *servlet* untuk aplikasi Java EE. Komponen *web* dan kontainernya berjalan pada *server* Java EE.
4. **Kontainer Aplikasi Klien.** Mengatur eksekusi komponen aplikasi klien. Aplikasi klien dan kontainernya berjalan pada komputer klien.
5. **Kontainer Applet.** Mengatur aksekusi applet. Kontainer *applet* terdiri dari sebuah *web browser* dan Java *plug-in* yang berada pada komputer klien.

2.3.3 *Servlet*

Servlet adalah kelas dalam bahasa pemrograman Java yang digunakan untuk memperluas kemampuan *server* agar mudah diakses melalui model pemrograman *request-response*. Java menyediakan paket *javax.servlet* dan *javax.servlet.http* berisi kelas dan *interface* yang dapat digunakan untuk menulis *servlet*. Semua *servlet* harus mengimplementasikan *servlet interface* yang mendefinisikan metode daur hidupnya. *Servlet* pada umumnya digunakan ketika halaman *web* yang dikirimkan kepada klien memiliki teks dinamis yang lebih besar daripada teks statis.

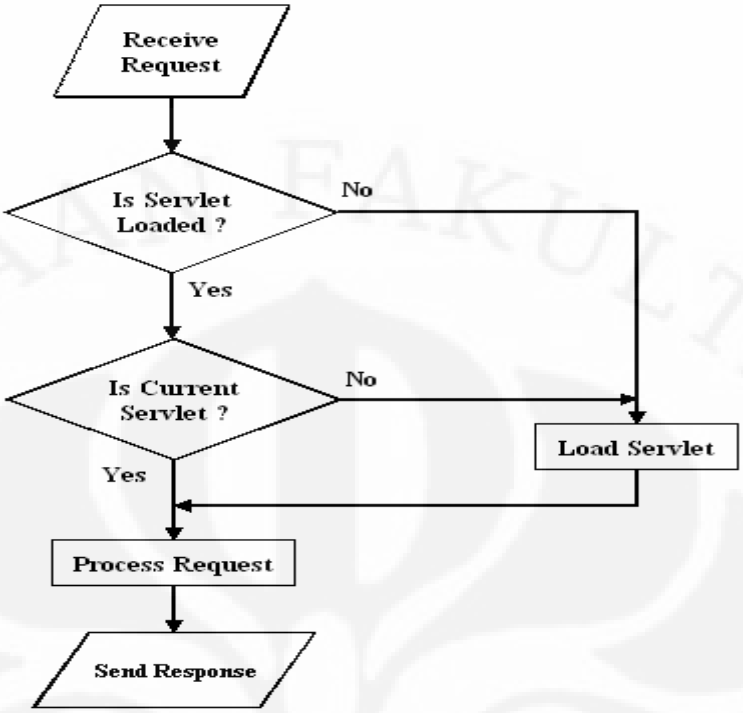
Servlet adalah kelas dalam Java yang diletakkan dan dijalankan melalui

server khusus, yang disebut kontainer JSP dan *servlet* (kontainer *web*). Kontainer biasanya sudah terintergrasi dengan *Sun Java System Application Server* (*wwws.sun.com/software/products/appsrvr/home_appsrvr.html*), *Microsoft's Internet Information Services* (*IIS*) (*www.microsoft.com/iis*), *Apache HTTP Server* (*httpd.apache.org*), *BEA's WebLogic application server* (*www.bea.com/products/weblogic/server/index.shtml*), *IBM's WebSphere application server* (*www-3.ibm.com/software/webservers/appserv/*) *and the World Wide Web Consortium's Jigsaw Web server* (*www.w3.org/Jigsaw/*).



Gambar 2.6 Arsitektur *servlet*

Sebuah klien mengirimkan *HTTP request* kepada *server* (Lihat Gambar 2.6), yang meminta *servlet* tertentu untuk pertama kali. Kontainer *servlet* menerima *request* dan mengirimkannya kepada *servlet* yang bersangkutan. Kemudian *servlet* memproses *request* tersebut sesuai dengan keperluan klien. Bila diperlukan, *servlet* akan menghubungi *database* atau komponen lainnya, seperti JSP atau *Enterprise JavaBean*. Setelah proses selesai, servlet mengembalikan hasilnya kepada klien dan biasanya dalam bentuk HTML, xHTML atau XML yang akan ditampilkan dalam browser klien. Setelah itu, *servlet* tinggal di dalam memori sambil menantikan lain *request* dan tidak akan dikosongkan dari memori kecuali jika kontainer *servlet* mendeteksi kekurangan memori. Setiap kali *servlet* di-*request* oleh klien, kontainer *servlet* membandingkan catatan waktu (*time stamp*) *servlet* yang berada dimemori dengan kelas *servlet*. Jika kelas *servlet* terjadi perubahan, *servlet* di-*reload* lagi ke dalam memori. Cara ini membuat pemrogram tidak harus menyalakan kembali kontainer *servlet* setiap kali terjadi pembaharuan terhadap *servlet* [8]. Lihat Gambar 2.7.



Gambar 2.7 Alur kerja *servlet*

2.3.4 *JavaServer Pages* (*JSP*)

JavaServer Pages™ (*JSP*™) adalah teknologi yang menyederhanakan kode *servlet* kedalam dokumen berbasis teks, sehingga memudahkan pemrogram non-Java menggunakan menggunakannya. *JavaServer Pages* berisi dua tipe teks, yaitu: statis data dan elemen JSP. Statik teks berisi berbagai format teks, seperti: HTML, SVP, WML dan XML. Sedangkan elemen JSP adalah bagian yang menentukan bagaimana sebuah halaman *web* bersifat dinamis.

Ada empat komponen kunci JSP: *directives*, *action*, *scripting elements* dan *tag libraries* [9]. *Directives* Adalah pesan kepada kontainer JSP, komponen *server* yang mengeksekusi JSP sehingga memungkinkan pemrogram untuk menentukan *setting* halaman, meliputi isi dari sumber daya lainnya dan untuk menetapkan *tag libraries* umum yang digunakan di dalam JSP. *Action* mengikat unit fungsional di dalam *predefined tags*, kemudian pemrogram dapat melekatkannya pada halaman JSP. *Action* sering dijalankan berdasarkan pada informasi yang dikirim kepada *server* sebagai bagian dari permintaan klien. *Action* juga dapat menciptakan Objek Java untuk digunakan di *scriptlets* JSP. *Scripting elements* memungkinkan pemrogram untuk menyisipkan Kode *Java* yang saling berhubungan dengan

komponen pada suatu JSP (atau Komponen aplikasi *web* lain) untuk memproses permintaan. *Scriptlets*, salah satu bentuk *scripting elements*, berisi fragmen kode yang mampu merespon permintaan klien. *Tag libraries* merupakan perluasan penggunaan *tag* yang memungkinkan para pemrogram untuk menciptakan *tag* khusus.

Markup language pada JSP yang disebut *fixed-template data* atau *fixed-template text*, sering membantu pemrogram dalam memutuskan apakah menggunakan JSP atau *servlet* [9]. Pemrogram sebaiknya menggunakan JSP ketika konten yang banyak dikirim ke klien adalah *fixed-template* data dan sedikit atau tidak ada konten dinamis yang dihasilkan oleh kode Java dan menggunakan *servlet* jika sedikit konten statisnya.

Ketika sebuah halaman JSP menerima *request* pertama dari klien, maka *server* yang mempunyai kontainer JSP, mengubah JSP kedalam bentuk *servlet* yang akan menangani *request* saat ini dan yang akan datang terhadap halaman JSP tersebut. JSP kontainer kemudian meletakkan pernyataan Java yang mengimplementasikan respon JSP ke dalam *method _jspService* pada waktu translasi JSP ke *servlet*. Jika *translasi berjalan baik*, kontainer JSP memanggil *_jspServices* agar menangani *request*. JSP dapat merespon secara langsung atau memanggil komponen *web* lain untuk membantu memproses *request*. Kesalahan pada waktu translasi disebut *translation-time errors* dan kesalahan pada waktu memproses *request* disebut *request-time errors* [9].

2.3.5 *Enterprise JavaBean* (*EJB*)

Enterprise bean adalah adalah komponen perangkat lunak yang berjalan pada sisi *server* (*server-side component*) yang disebarkan (*deployed*) dalam sebuah lingkungan komponen terdistribusi bertingkat (*distributed multi-tier environment*). *Enterprise bean* ditulis dengan menggunakan EJB API yang telah tersedia di Java, paket *javax.ejb*, dan dijalankan dalam sebuah kontainer EJB, dimana biasanya kontainer EJB ini terintegrasi di dalam *application server*. EJB sangat cocok diterapkan pada aplikasi *scalable*, integritas data dan mempunyai banyak macam klien [10].

2.3.5.1 Keuntungan EJB

Ada beberapa keuntungan yang bisa di dapat jika aplikasi komponen bertingkat (*multi-tier*) menggunakan EJB, seperti tertulis dibawah ini [11] :

1. Kontainer EJB menyediakan *system-level services* ke *enterprise bean*, seperti *transaction management* dan keamanan, sehingga pemrogram *bean* dapat berkonsentrasi dalam memecahkan aplikasi bisnis.
2. Aplikasi logika bisnis berada pada *bean*, sehingga pemrogram komponen klien dapat fokus pada pembuatan GUI yang yang baik. Pemrogram komponen klien tidak perlu menulis kode rumit yang menerapkan aturan logika bisnis atau mengakses *database*. Sebagai hasilnya, komponen klien menjadi lebih ringan, sangat manfaat untuk klien yang berjalan peralatan dengan kemampuan terbatas, seperti telepon genggam, PDA, atau PDA *phones*.
3. *Enterprise bean* bersifat *portable*, sehingga aplikasi *assembler* dapat membangun aplikasi baru dari *bean* yang ada. Aplikasi ini dapat berjalan pada *server* Java EE apapun dengan ketentuan bahwa *bean* dan aplikasi baru tersebut menggunakan Java API standar.

2.3.5.2 Kontainer EJB

Kontainer EJB bertanggung jawab pada pengaturan *enterprise bean* dengan menawarkan berbagai macam layanan. Namun demikian layanan tersebut baru bisa digunakan jika diimplementasikan dalam kode yang dimengerti oleh kontainer. Ini berlaku juga untuk komponen aplikasi klien terhadap kontainernya. *Bean* dapat memberitahukan kontainer secara implisit informasi apa yang akan dibutuhkan dalam sebuah *deployment descriptor* atau melalui *deployment annotations*. Berikut ini merupakan beberapa layanan yang ditawarkan oleh kontainer EJB [12]:

1. **Keamanan**. Keamanan menjadi perhatian utama dalam aplikasi komponen bertingkat (*multi-tier application*). Java SE telah menerapkan lingkungan yang aman dengan melakukan verifikasi dan otorisasi dalam mengakses kode Java, misalnya penerapan *method set* dan *get*, melalui EJB sistem keamanan bertambah dengan tersedianya *security* API.

2. **Pengaturan *transaction***. Manajemen yang mengatur proses suatu perangkat lunak dalam sebuah komponen dengan perangkat lunak dalam komponen lainnya, sehingga hanya mengizinkan kedua proses itu berjalan atau keduanya tidak berjalan sama sekali. Contohnya ketika seseorang mengambil uang dari sebuah rekening di bank, kemudian mengirim uang tersebut ke rekening ke bank yang sama atau berbeda, maka kedua proses tersebut harus berjalan atau kedua-duanya tidak berjalan. Hal ini diatur oleh *transaction management*. Layanan *transaction* disediakan dalam *Java Transaction API (JTA)*.
3. **Pengaturan sumber daya dan daur hidup**. Kontainer EJB mengatur sumber daya (*resources*) seperti *sockets, threads,* dan koneksi *database* sebagai sebuah *bean*. Selain itu, kontainer juga mengatur daur hidup (*life cycle*) *enterprises bean*. Misalnya, kontainer membuat objek *bean* dan me-nonaktifkan atau menghancurkan jika tidak dibutuhkan.
4. **Akses tersembunyi**. Klien yang dieksekusi oleh *java virtual machine* yang tersembunyi (*remote*) dapat memanggil *method* pada sebuah *enterprise bean* tanpa perlu penulisan kode untuk akses tersembunyi (*remote accessibility*).
5. **Mendukung *concurrent request***. *Concurrent request* berarti *request* yang terjadi secara bersamaan. Kontainer EJB memungkinkan melakukan *concurrent request* tanpa penulisan kode *multithread*. Contohnya ketika banyak klien mengakses sebuah *method* pada objek *bean,* kontainer akan menjadikan panggilan terhadap *method* tersebut berseri (*serialize*), sehingga hanya satu klien yang bisa memanggil *bean* pada waktu yang bersamaan, dan panggilan yang lain dilaksanakan ketika panggilan pertama selesai.

2.3.5.3 Jenis-jenis Bean

Ada tiga jenis *bean* (Lihat Gambar 2.8), yaitu : *session bean, entity bean* dan *message-driven bean* [13]. Penerapannya dapat disesuaikan dengan kebutuhan pemrogram.

1. ***Session bean***. Suatu *session bean* menggambarkan klien tunggal *application server*. Komponen klien memanggil *method* dalam *session bean* terlebih dahulu jika mengakses aplikasi yang berjalan pada *server*. *Session bean* melakukan kerja untuk kliennya, melindungi klien dari kompleksitas eksekusi

logika bisnis di dalam *server*. Data dari *session bean* tidak disimpan didalam *database*. Daur hidup *session bean* berakhir jika selesai melayani kliennya.

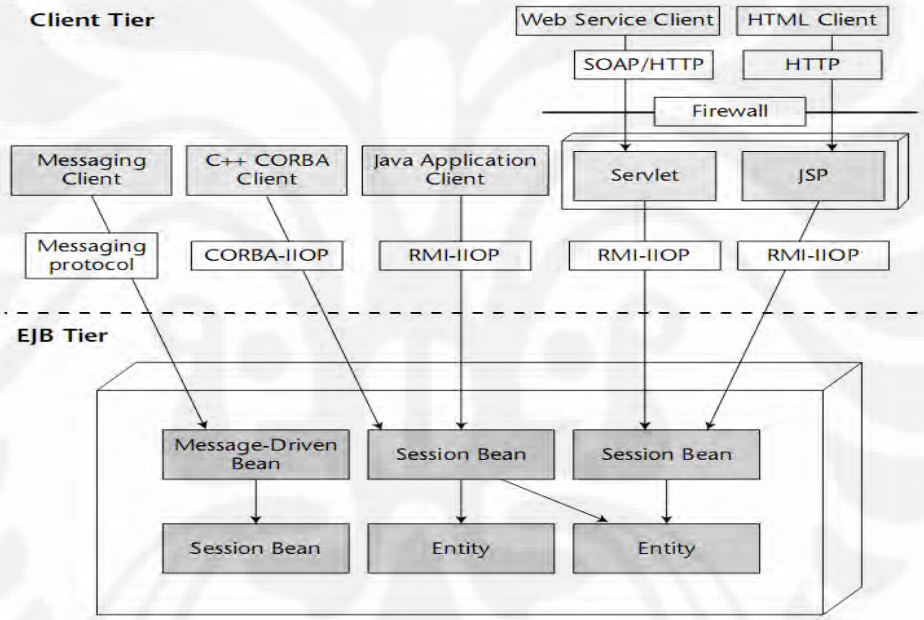
Sesion bean terbagi menjadi dua kategori, yaitu : *stateless session bean* dan *stateful session bean*. *Stateless session bean* adalah sebuah *bean* yang hanya berkomunikasi atau mempunyai daur hidup selama klien melakukan pemanggilan *method* yang berada pada *bean* yang bersangkutan. Kata *stateless* menunjukkan bahwa *session bean* ini tidak mempertahankan status komunikasi setelah proses pemanggilan *method bean* tersebut selesai. Setelah pemanggilan *method*, kontainer dapat memilih menghancurkan, membuatnya kembali dengan menghapus semua status (informasi) yang berhubungan dengan pemanggilan *method stateless session bean* sebelumnya atau menggunakannya kembali untuk klien yang ingin menggunakan kelas *session bean* tersebut.

Stateful session bean adalah sebuah *bean* yang didesain untuk melayani banyak *request* dan *transaction* dengan tetap menjaga status (informasi) kliennya (*retain state*). Jika keadaan sebuah *stateful session bean* dirubah selama pemanggilan *method*, status tersebut masih akan ada sampai saat klien yang sama melakukan pemanggilan *method* berikutnya. hal ini tentu sangat menyulitkan kontainer dalam menyiapkan (*pool*) dan menandai *bean* secara dinamis agar bisa menangani *request* klien dalam jumlah besar, karena memerlukan banyak memori untuk menyimpan status tersebut. Kejadian ini mirip dengan sebuah komputer yang sedang menjalankan banyak aplikasi secara bersamaan. Ada aplikasi yang aktif dipakai ada juga yang yang pasif, keduanya tetap membutuhkan memori. Kondisi ini tentu saja akan menghabiskan sumber daya memori. Kontainer EJB mengambil paradigma komputer dalam menyelesaikan permasalahan ini. Objek *bean* yang tidak aktif ditaruh di dalam sebuah *hard disk* atau *virtual memory* (*passivation*). Kemudian jika *method* dalam objek *bean* tersebut dipanggil oleh klien, objek *bean* tersebut diaktifkan kembali (*activation*).

2. ***Message-driven bean***. *Message-driven bean* mempunyai kemiripan dengan *session bean* dalam melakukan aksi, yaitu sama-sama pasangan klien-EJB. *Message-driven bean* dipanggil oleh kontainer ketika datang pesan atau ketika akhir dari layanan yang diberikan oleh *message-driven bean*. Namun demikian, klien tidak dapat mengkases *message-driven bean* secara langsung melalui

interface tetapi melalui sebuah mekanisme *messaging system*, seperti *Java Messaging Service (JMS). Message-driven bean* bersifat *stateless* dan hanya memproses satu pesan pada satu waktu yang sama.

3. ***Entity bean***. *Entity bean* adalah sebuah komponen data yang menyimpan data secara permanen pada sebuah media penyimpanan seperti *database*. *Entity bean*, seperti kelas Java yang lain, mempunyai parameter untuk menyimpan data dan *method* untuk melakukan operasi dari sebuah parameter. Hal ini berarti, *entity bean* merubah data menjadi objek Java kemudian memanipulasinya. *Entity bean* menggambarkan rekaman dan proses pengaturan tabel *database*, sehingga klien tidak perlu berhubungan langsung dengan *database server*.



Gambar 2.8 Macam-macam *bean* dan klien yang dapat ditangani

2.4 *WEB SERVICE*

Web service adalah aplikasi berskala besar (*enterprise application*) berbasis *web* yang menawarkan koleksi layanan yang dapat diakses oleh klien melalui pesan berbasis XML [14]. *Web service* memiliki antar muka (*interface*) yang didefiniskan, dideskripsikan dan dimengerti oleh XML dan juga mendukung interaksi langsung dengan *software* aplikasi yang juga pesan berbasis XML melalui sebuah protokol internet.

2.4.1 Standar *Web Service*

Standar *core* dan protokol aplikasi *web service* didefinisikan dan diatur oleh *Web Service Interoperability Organization* (*http://www.ws-i.org/*), *World Wide Web Consortium* (*http://www.w3.org/2002/ws/*) dan *Organization for the Advancement of Structured Information Standards* (*http://www.oasis-open.org/who/*). Berikut ini merupakan standar yang digunakan pada *web service*:

***Simple Object Access Protocol* (*SOAP*)**. SOAP adalah sebuah *mark-up language* berbasis XML untuk pertukaran pesan antar aplikasi. SOAP seperti sebuah amplop yang digunakan untuk pertukaran data didalam jaringan. SOAP mendefinisikan empat aspek didalam komunikasi: *Message envelope, encoding, RPC call convention* dan bagaimana menyatukan sebuah pesan di dalam protokol tingkat *transport*. Sebuah pesan SOAP terdiri dari SOAP *envelop* dan *attachments* sebagai opsional. SOAP *envelop* tersusun dari SOAP *header* dan SOAP *body*. Sedangkan SOAP *attachment* mengizinkan data non-XML untuk dimasukkan kedalam SOAP *message*, di-*encoded* dan diletakkan kedalam SOAP *message* menggunakan MIME *multipart*.

***Extensible Markup Language* (*XML*).** XML adalah is a *mark-up language* yang berisi informasi yang terstruktur. XML menggambarkan sebuah kelas dari sebuah objek data yang disebut dokumen XML dan secara parsial menggambarkan juga kebiasaan program komputer yang memprosesnya [15]. XML menjadi bagian dalam *web service* baik dalam mendeskripsikan layanan kepada klien atau sebagai media untuk membawa informasi melalui pesan XML. Sebuah dokumen XML, sama halnya dengan HTML, terdiri dari *tag* atau elemen yang didefinisikan dengan kurung siku (< dan >). XML *tag* bersifat *extensible*, sehingga memungkinkan pemrogram untuk menulis XML *tag* sendiri dalam mendeskripsikan sebuah isi (*content*) agar sesuai dengan keinginannya. XML *tag* didefinisikan menggunakan XML *schema language*, yaitu skema yang mendefinisikan struktur dokumen XML dan juga digunakan untuk membatasi isi (*content*) dokumen XML pada sebuah element, atribut dan nilai tertentu.

***Web Service Description Language* (*WSDL*)**. WSDL.adalah sebuah XML *based language* untuk mendeskripsikan XML. WSDL menyediakan layanan yang mendeskripsikan *request* layanan dengan menggunakan protokol-protokol yang

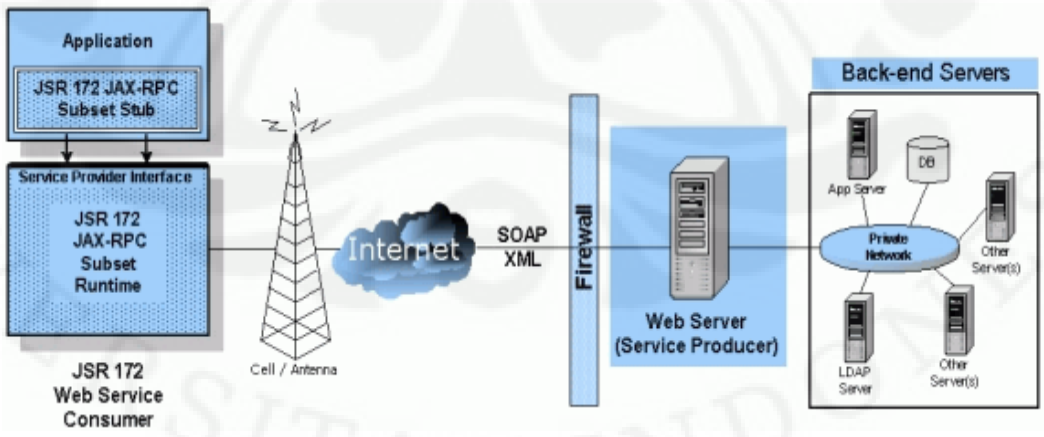
berbeda, melakukan *encoding* dan juga memfasilitasi komunikasi antar aplikasi. WSDL mendeskripsikan apa yang akan dilakukan oleh *web service*, bagaimana menemukannya dan mengoperasikannya.

***Universal Description, Discovery and Integration* (*UDDI*)**. UDDI adalah sebuah *service registry* bagi pengalokasian *web service*. UDDI mengkombinasikan SOAP dan WSDL untuk pembentukan sebuah *registry* API bagi pendaftaran dan pengenalan sebuah layanan. UDDI juga menyediakan sebuah area umum dimana sebuah organisasi dapat mengiklankan keberadaan mereka dan layanan yang mereka berikan. UDDI menjadi *framework* yang mendefinisikan sebuah XML *based registry* dimana sebuah organisasi dapat melakukan *upload* informasi mengenai layanan yang mereka berikan. XML *based registry* berisi nama-nama dari organisasi tersebut, beserta layanan dan deskripsi dari layanan yang mereka berikan.

2.4.2 Arsitektur *Web Service* pada Standar Java ME.

Arsitektur *web service* mempunyai tiga elemen (lihat Gambar 2.9) :

1. Aplikasi klien *web service* yang berada pada perangkat yang mendukung *web service*.
2. Jaringan *wireless* dan internet serta pendukungnya.
3. *Web server* yang bertindak sebagai penyedia layanan dan aplikasi *back-end*.



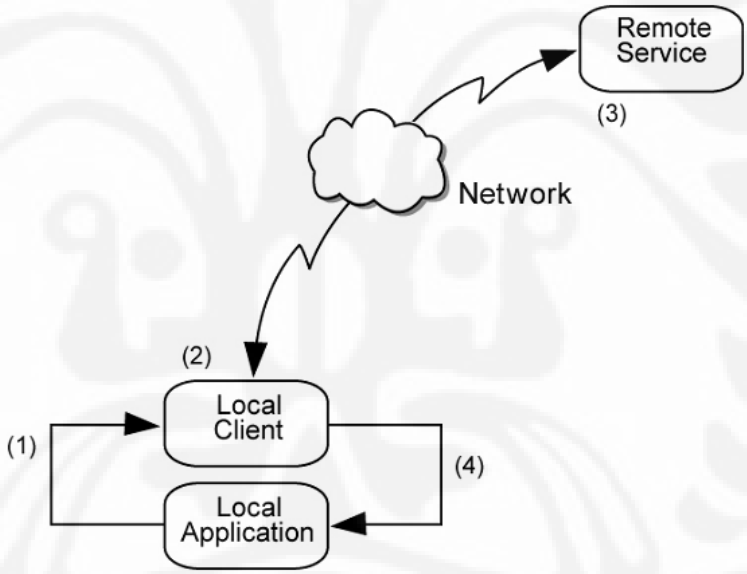
Gambar 2.9 Arsitektur *Web service* pada Java EE [16].

2.4.3 Interaksi Klien dengan *Web Service*.

Berikut ini akan dijelaskan bagaimana interaksi antara klien *web service* dan

remote web service (Lihat Gambar 2.10):

1. Sebuah aplikasi lokal melakukan pemanggilan prosedur ke komponen klien lokal, sebuah komponen yang dibuat menggunakan file WSDL yang diperoleh dari sebuah *server web service.*
2. Klien lokal membuat kanal komunikasi HTTP dengan *server web service*, dan berinteraksi menggunakan SOAP.
3. *Server web service* menerima panggilan prosedur, melakukan proses, dan mengembalikan respon kepada klien lokal melalui kanal komunikasi HTTP menggunakan SOAP.
4. Komponen klien lokal mengembalikan respon pada lokal aplikasi. Transaksi terjadi seolah-olah hanya pada perangkat klien.



Gambar 2.10 Interaksi antara klien dengan *Web services* [17].

2.5 *GLOBAL PACKET RADIO SERVICE* (*GPRS*)

GPRS merupakan layanan pengiriman data berbasis paket yang ditambahkan pada jaringan *Global Systems for Mobile Communications (GSM)*. GPRS dikembangkan dengan tujuan untuk memenuhi kebutuhan akan layanan paket data wireless yang merupakan dampak dari pesatnya pertumbuhan internet. Teknologi GPRS memungkinkan proses pengiriman data dengan kecepatan

hingga mencapai 171.2 kbps [19] dan lebih murah jika dibandingkan dengan *circuit-switched* data pada GSM.

Beberapa contoh aplikasi yang menggunakan teknologi GPRS diantaranya adalah *chat*, pengiriman data/informasi berbasis teks dan visual, pengiriman data gambar, *web browsing*, transfer file diantaranya *MMS* (*Multimedia Message Service*), aplikasi *e-mail* korporat dan internet serta SMS.

GPRS mampu menghasilkan *bit rate*, secara teoritis, sekitar 171 kbps. Nilai ini dicapai dengan pengaturan algoritma *coding scheme* dengan koreksi *error* terendah dan delapan *timeslot*. *Coding schemes* [19] dipengaruhi oleh kondisi *radio link* diantara *mobile station* dan BTS. Ada empat macam *coding schemes* yaitu: *CS-1, CS-2, CS-3* dan *CS-4*. CS-1 digunakan untuk kondisi *radio link* paling buruk karena mempunyai koreksi terhadap kesalahan paling baik. Sementara CS-4 tidak mempunyai koreksi terhadap kesalahan sehingga hanya dapat digunakan pada kondisi *radio link* yang sangat baik.

Tabel 2.1 Kelas GPRS *multislot* [19]

Multislot GPRS class	Downlink slots	Uplink slots	Active slots
1	1	1	2
2	2	1	3
3	2	2	3
4	3	1	4
5	2	2	4
6	3	2	4
7	3	3	4
8	4	1	5
9	3	2	5
10	4	2	5
11	4	3	5
12	4	4	5

Kelas *multislot* (Tabel 2.1) divais menunjukkan banyaknya *timeslot* yang bisa digunakan untuk *downstream, upstream* dan totalnya. Contohnya *mobile station* dengan *GPRS* kelas 6 mempunyai maksimum tiga *slot* untuk *downlink* dan maksimum dua slot untuk *uplink*. Lihat Tabel 2.2. Sehingga maksimum total *slot* yang bisa digunakan adalah lima. Kapasitas dari tiap *slot* tergantung pada koding yang digunakan. Sehingga penggunaan *slot* yang semakin banyak akan membutuhkan kemampuan *processing* dan konsumsi daya yang lebih tinggi.

Tabel 2.2 Maksimum *bit rate* GPRS secara teoritis (dalam kbps)[19]

Timeslots	1 TS	2 TS	3 TS	4 TS	5 TS	6 TS	7 TS	8 TS
CS-1	9.05	18.10	27.15	36.20	45.25	54.30	63.35	72.40
CS-2	13.40	26.80	40.20	53.60	67.00	80.40	93.80	107.20
CS-3	15.60	31.20	46.80	62.40	78.00	93.60	109.20	124.80
CS-4	21.40	42.80	64.20	85.60	107.00	128.40	149.80	171.20

2.6 *MOBILE TERMINAL*

Harga perangkat yang semakin murah dan permintaan dari masyarakat membuat *mobile* divais kini tersebar diberbagai lapisan masyarakat.

2.6.1 Jenis-jenis *Mobile* Divais

Berikut ini penjelasan berbagai *mobile* divais yang ada dipasaran saat ini:

Laptop. Mempunyai kemampuan sekelas komputer *desktop*, walaupun mempunyai ukuran lebih kecil. Konektivitas laptop mutakhir biasanya sangat baik, dilengkapi *infrared port*, *Bluetooth*, Wi-Fi a/b/g. Bahkan untuk beberapa tahun kedepan, sudah terintegrasi dengan WiMAX.

Tablet PC. Tablet PC adalah titik kulminasi dalam menyusutkan *laptop*. Kemampuan pengolah data hampir sama dengan *laptop*. Konektivitas sama dengan *laptop*, namun dengan harga relatif lebih murah. Tablet PC mempunyai inputan *digitizer* atau *touch screen*. Kelemahan biasanya pada daya tahan baterei yang tidak terlalu lama.

Personal Digital Assistant (PDA). Mempunyai ukuran kecil mampu mengolah berbagai jenis file *multimedia*. Model terbaru mempunyai layar dengan kedalaman warna lebih dari 262.000 warna dan mempunyai kemampuan *handwritten* teks. Sistem operasi biasanya menggunakan *Palm OS* Dan *Microsoft Pocket PC*.

Telepon Selular. *Mobile* divais yang paling banyak digunakan saat ini dengan kemampuan komunikasi suara dan pengiriman dan menerima pesan teks (SMS) dan *multimedia* (MMS). Kelemahan perangkat ini adalah kecilnya kapasitas memori dan kemampuan pemrosesnya. Untuk pengiriman data, perangkat ini dilengkapi dengan *WAP (Wireless Application Protocol)* dan GPRS. Konvergensi komunikasi suara dan data semakin baik dengan hadirnya teknologi

3G.

SmartPhone. Merupakan perpaduan antara telepon selular dan PDA. Sistem operasi yang biasa digunakan adalah *Symbian OS* dan *Windows Mobile*. Kemampuan *smartphone* hanya sedikit dibawah PDA namun dengan bentuk seperti telepon selular.

2.6.2 Sistem Operasi

Selain heterogen dalam hal *hardware*, *mobile* divais juga heterogen dalam hal *software*, dalam hal ini sistem operasi. Namun demikian, pada bagian ini hanya dibahas sistem operasi yang popular saja.

Microsoft Windows*.** Tidak seperti pada industri komputer, *Microsoft Corp.* tidak lagi menguasai sistem operasi pada divais *mobile*. *Microsoft Corp.* mengeluarkan dua varian untuk sistem operasi *mobile*, yaitu: *Windows CE* dan *Windows Mobile*. Arsitektur *32-bit* yang dirancang untuk memenuhi kebutuhan suatu jangkauan luas divais yang cerdas. *Windows Mobile* nampaknya terus berkembang seiring dengan perkembangan divais yang mendukungnya. Sampai saat ini (*Windows Mobile 6.0)* telah mendukung berbagai aplikasi komputer *desktop* mutakhir, seperti *Microsoft Office, Windows Live Messenger,* sikronisasi dengan *Windows Vista, instant messaging (IM), Microsoft .NET Compact Framework and Microsoft SQL Server™,* fasilitas VoIP dan *blogging.**

Palm OS adalah sistem operasi komputer yang menyediakan suatu *platform* perangkat lunak untuk *PDA* yang dibuat oleh *Palm Computing*. Meskipun *Palm OS* bersifat *multitasking*, namun satu program aplikasi harus selesai sebelum aplikasi lain berjalan. Aplikasi dan *database* disimpan dalam dalam penyimpanan permanen RAM yang tidak bisa digunakan kembali seperti dinamik RAM. *Palm OS* membagi aplikasi kedalam *runnable code* dan berbagai tipe data elemen seperti elemen *interface* dan *icon*. data elemen ini dapat dengan mudah dirubah tanpa perlu menulis ulang kode. Fitur pendukung untuk pengembangan *Palm OS* menyediakan: *Software Development Kit (SDK)* yang menyediakan *application programming interfaces (APIs)* untuk pengembangan aplikasi. Selain itu ada *Conduit Development Kit (CDK)* untuk melakukan pertukaran dan sinkronisasi data dengan aplikasi pada komputer *desktop*.

Symbian OS adalah sistem operasi yang didesain untuk *computer-telephones* dengan bentuk kecil dan *portable* yang mempunyai kemampuan koneksi nirkabel. *Symbian OS* mengandalkan sistem komputasi *32-bit*, *multitasking* dan *pen-based* GUI.

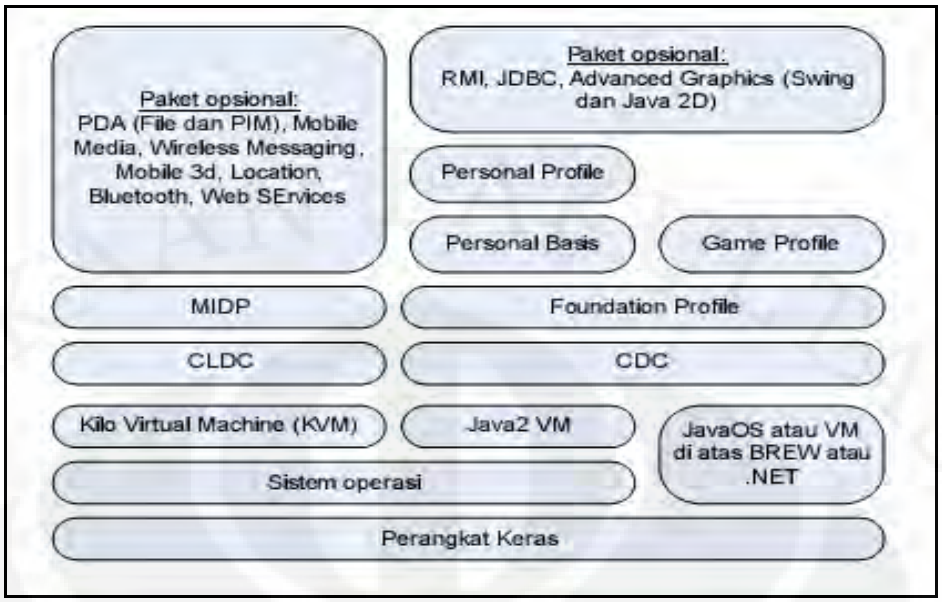
Symbian OS sudah mendukung beberapa fitur pesan internet, seperti: email menggunakan POP3, IMAP4, SMTP, MHTML dengan protokol TCP/IP dan WAP. *Symbian OS* juga menyediakan *development kits* untuk C++, OPL, dan Java. Selain itu disediakan juga *middleware* untuk komunikasi, data managemen dan grafik, level rendah GUI *framework*, dan *application engines*.

2.6.3 Konfigurasi

Konfigurasi berkaitan dengan fitur-fitur bahasa *Java* dan *library* yang bisa dijalankan oleh *Java Virtual Machine* (*JVM*). Saat ini, terdapat dua jenis Konfigurasi yaitu *Connected Device Configuration* (CDC) dan *Connected Limited Device Configuration* (CLDC).

1. Karakteristik perangkat dengan konfigurasi CDC:
    a. Memori minimal 512 KB (*kilobyte*) untuk menjalankan Java
    b. Alokasi memori untuk runtime minimal 256 *kilobyte*
    c. Konektivitas jaringan, mungkin bandwidth tinggi dan persisten
2. Karakteristik perangkat dengan konfigurasi CLDC:
    a. Memori minimal 128 *kilobyte* untuk menjalankan Java
    b. Alokasi memori untuk runtime minimal 32 *kilobyte*
    c. Antar muka pengguna yang terbatas
    d. Sumber daya rendah, umumnya daya diperoleh dari baterai
    e. Konektivitas jaringan, umumnya nirkabel dengan bandwidth rendah

Secara lebih jelas, dukungan konfigurasi ini dapat dilihat pada Gambar 2.11



Gambar 2.11 Arsitektur Java ME [20]

BAB III
DESAIN DAN IMLEMENTASI SMILE

3.1 POKOK PERMASALAHAN

SMiLE (*System of Mobile Learning Environment*) merupakan sebuah sistem yang dibuat untuk mengimplementasikan konsep *mobile learning*. Visi dari SMiLE adalah *Smart Client for Learning Anywhere, Anytime and Anyhow*. SMiLE mengimplementasikan arsitektur *4-tier* (empat tingkat). Pada komponen klien (*client-tier*) dibuat program berbasis Java ME yang menjadi antar muka antara pengguna (*user*) dan SMiLE yaitu *MIDlet* dan klien *web service*. Sedangkan *browser* HTML/xHTML berasal dari *vendor* perangkat klien. Program ini dijalankan dalam kotainer Java yang terintegrasi pada perangkat klien, seperti : PDA *phone*, PDA, *mobile phone* dan *smart phone*. Pada sisi *server* dibangun tiga *tier*, yaitu komponen *web* yang terdiri dari aplikasi *web* berbasis *servlet* dan JSP. Komponen bisnis dibuat berdasarkan *Enterprise JavaBean 3.0* dan komponen *enterprise information system* (*Derby Database*). Lihat juga Sub-sub Bab 2.2.3 kontainer Java EE.

SMiLE cukup sulit untuk diimplementasikan. Hal ini disebabkan perangkat klien yang digunakan sangat heterogen, dari sistem operasi sampai kemampuan klien seperti: ketersediaan daya, luas area visualisasi, *processor*, *input device*, media penyimpanan dan *memory*. Keterbatasan ini tentu menuntut pemrogram (pengembang perangkat lunak) berfikir beda tentang *user interface* pada klien, kemampuan komputasi, pengaturan memori, dan penanganan kesalahan (*exception handling*) jika dibandingkan dengan pembuatan program untuk klien untuk komputer *desktop*.

Selain itu ketersediaan jaringan komunikasi, *bandwidth* dan *realibility*, masih sangat buruk dan kalaupun ada, masih mahal serta belum umum digunakan dimasyarakat. Klien yang relatif selalu bergerak (*high mobility*) juga menambah resiko untuk mengurangi bahkan menghilangkan kemampuan layanan koneksi.

3.2 SOLUSI PERMASALAHAN

Perbedaan *platform* pada klien dan *server*, diatasi dengan penggunaan *web service*. *Web service* adalah sebuah aplikasi yang berjalan tanpa dipengaruhi oleh *platform* apa dibuat dan arsitektur seperti apa dia di-*deploy*. *Web service* membuat sebuah lingkungan dimana klien dapat menghubunginya melalui WSDL dan UDDI kemudiaan dijalankan pada berbagai jenis peralatan. Penggunaan *web service* juga dapat mengurangi kesalahan yang ditimbulkan akibat penggunaan jaringan *wireless*, karena semua data yang dibutuhkan oleh klien dikirimkan ketika klien meminta *request* melalui dan diterima oleh klien generator (*stub generator*), *stub generator*-lah yang kemudian berinteraksi dengan klien. Sehingga seolah-olah *transaction* terjadi hanya pada *localhost*. Berkurangnya komunikasi antara klien dan *server* tentu akan mengurangi kemungkinan kesalahan (*error*) ketika terjadi pengiriman data. Kemungkinan kesalahan (*error*) ini semakin diperkecil jika *web service* langsung berhubungan dengan *enterprise bean*, karena kontainer EJB mempunyai layanan pengaturan *transaction* (Lihat Sub-sub-sub Bab 2.2.6.2 kontainer EJB).

Penggunaan *amplop* SOAP yang berbasis XML juga menawarkan solusi lain bagi klien SMiLE yang mempunyai kemampuan terbatas, karena XML membuat data yang diterima klien bersifat *portable*, sehingga data berbasis XML dapat diterima oleh berbagai jenis klien.

3.3 PROTOKOL KOMUNIKASI

Penggunaan protokol aplikasi HTTP pada *web service* juga berdampak baik pada proses pengiriman pesan *enterprise*, yaitu pesan penting yang tidak boleh sedikitpun hilang bagiannya, seperti transaksi pada *bank*. Protokol HTTP yang bersifat *connection-oriented* mengharuskan inisialisasi sebelum melakukan pertukaran informasi untuk memastikan data sampai pada tujuan. Sehingga penggunaan protokol HTTP (*connection-oriented*) pada jaringan nirkabel, yang memang tidak *reliable* (terutama GPRS), sangat tepat. Permasalahan yang akan dihadapi adalah keterbatasan *bandwidth*. Proses inisialisasi pada protokol berbasis TCP/IP tentu berbeda dengan inisialisasi pada jaringan 2G GSM meskipun disini SMiLE menggunakan jaringan GPRS. Pada GSM proses inisialisai berada pada

line yang berbeda dengan *line* telepon, sehingga inisialisasi tidak mengganggu *line* telepon. Sementara pada protokol HTTP inisialisasi dilakukan pada *line* yang sama dengan *line* data, sehingga *bandwidth* untuk data juga digunakan untuk proses inisialisasi. Namun demikian, menurut penulis, ini masih lebih baik jika dibandingkan menggunakan protokol yang bersifat *connectionless*, yang mebutuhkan aplikasi tambahan agar bisa memastikan data sampai ketujuan karena hal ini dapat memperberat kerja klien.

3.4 ARSITEKTUR SMILE

SMiLE, seperti telah disebutkan sebelumnya, dibangun berdasarkan pendekatan konsep arsitektur komponen *multi-tier* yang dapat terdistribusi (*distributable multi-tier component*). Komponen-komponen tersebut adalah komponen klien, komponen *web*, komponen logika bisnis dan komponen *database*. Sedangkan pengertian dapat terdistribusi (*distributable*) berarti setiap komponen berdiri sendiri, dapat berjalan tanpa bergantung pada komponen lainnya. Komunikasi antar tiap komponen dilakukan dengan pemanggilan *method* pada kelas-kelas Java dan *service* berbasis pesan XML pada *web service*.

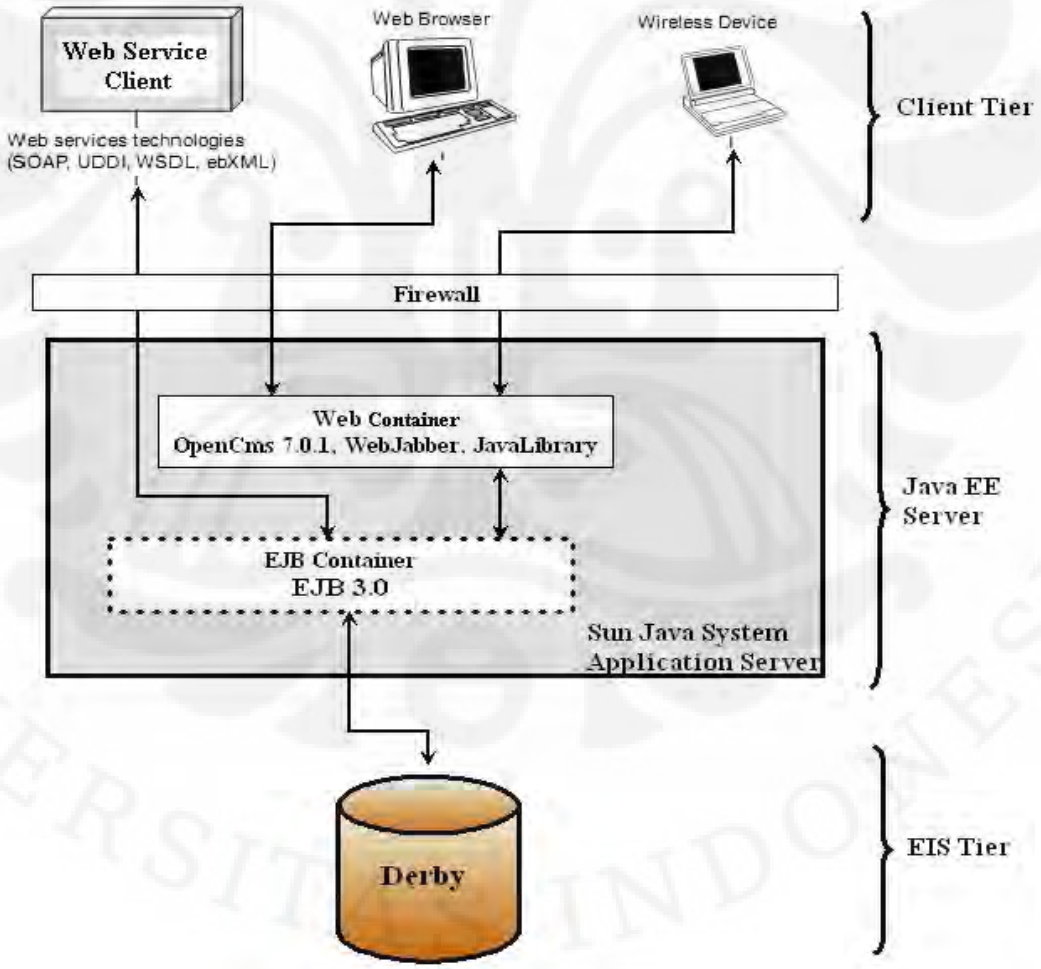
Tabel 3.1 Spesifikasi MIDP 2.0

Spesifikasi	MIDP 2.0
Display	96 x 54
display	1 bit
Bentuk pixel	Mendekati 1 : 1
Input	Keyboard, touch screen
Memori	**1.** 256 Kb non volatile **2.** 8 Kb non volatile data persistence yang dibuat oleh aplikasi **3.** 32 Kb memori non volatile
Multimedia	Memiliki kemampuan multimedia (suara dan video)

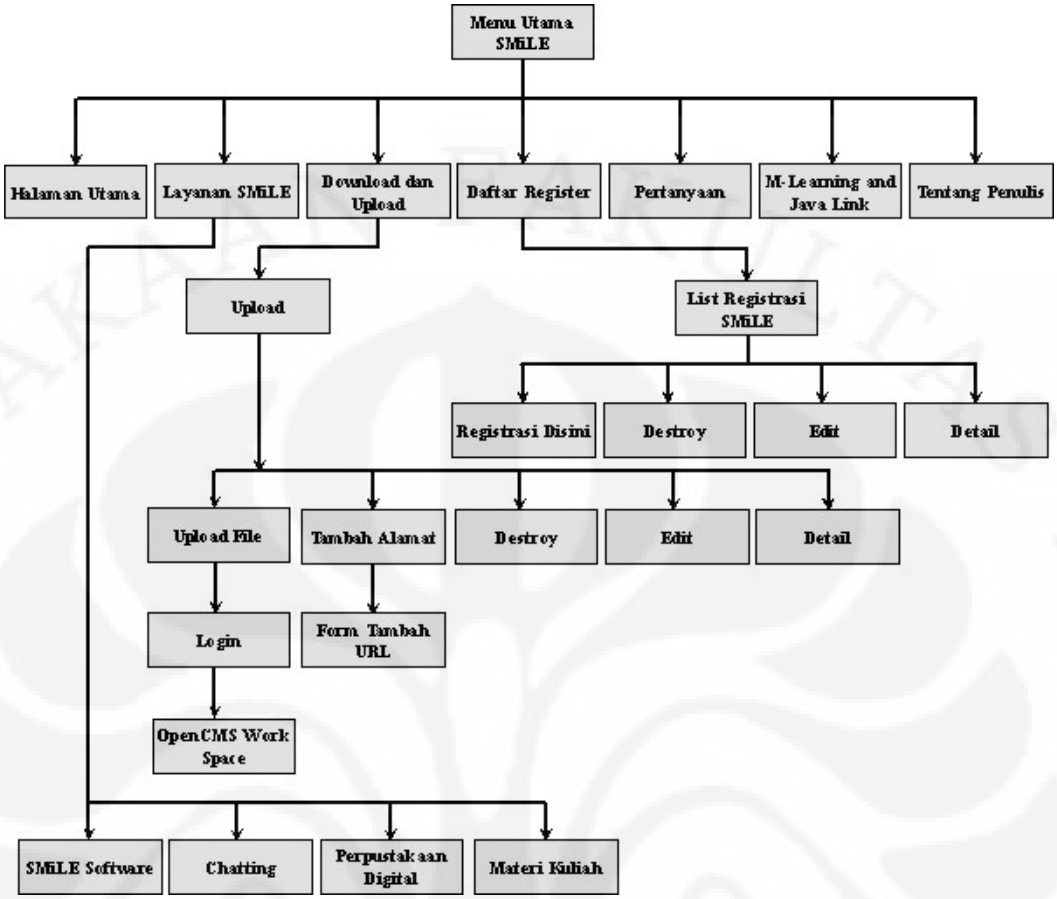
Komponen klien terdiri dari semua komputer *desktop* dan *mobile divice* : *mobile phone, smart phone,* PDA *phone* yang telah mendukung teknologi Java

ME dengan konfigurasi minimum *CLDC* (*Connected Limited Device Configuration*) dan sudah mendukung profil *MIDP* (*Mobile Information Device Profile*) 2.0, seperti ditunjukkan oleh Tabel 3.1.

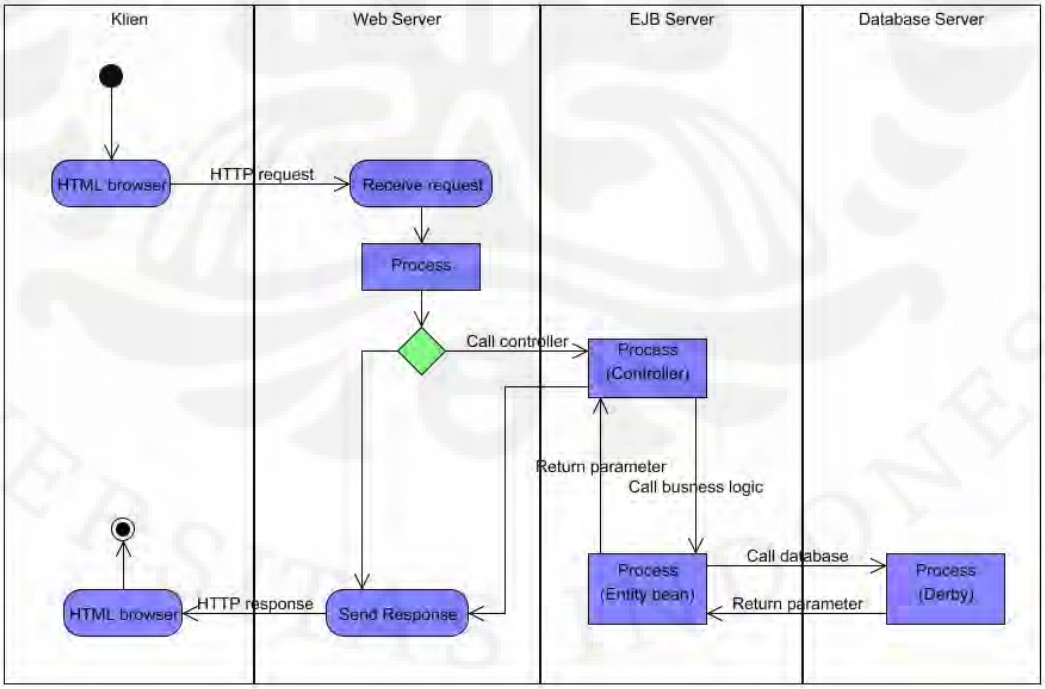
Komponen *web* yang berjalan pada kontainer *web* adalah JSP dan *servlet*. Pada komponen *web*, penulis mengintegrasikan aplikasi *chatting WebJabber* untuk memenuhi kebutuhan komunikasi yang intensif antara pelajar dan pengajar atau sesama pelajar, perpustakaan digital *JavaLibrary* dan *OpenCms 7.0.1* kedalam *web site* SMiLE. Pada komponen bisnis digunakan *Enterprise JavaBean* (*EJB*) 3.0. Sedangkan komponen *database* digunakan *Derby*, yang sudah terintegrasi dengan Java EE. Arsitektur komponen-komponen SMiLE dapat dilihat pada Gambar 3.1, sedangkan *flow chart, activity diagram,* dan *sequence diagram*-nya dapat dilihat pada Gambar 3.2, Gambar 3.3, Gambar 3.4.



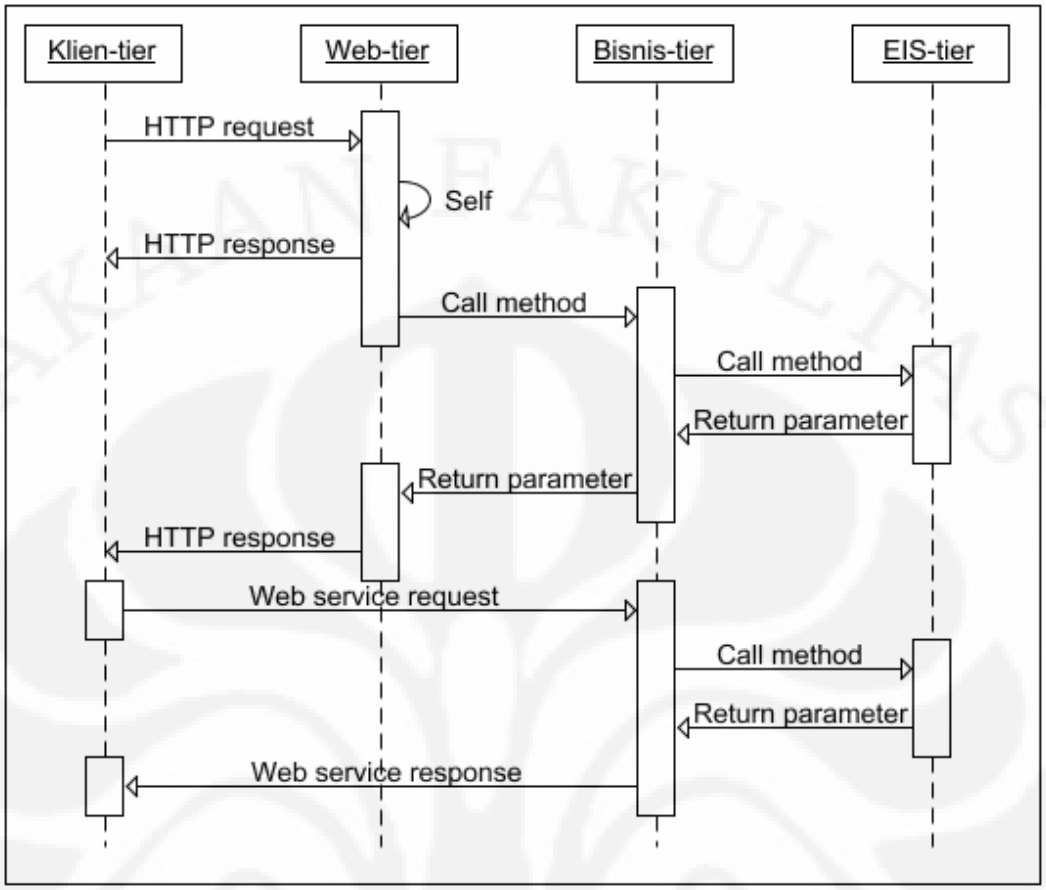
Gambar 3.1 Arsitektur SMiLE



Gambar 3.2 *Flow chart* SMiLE



Gambar 3.3 *Activity Diagram* SMiLE



Gambar 3.4 *Sequence Diagram* SMiLE

3.5 IMPLEMENTASI

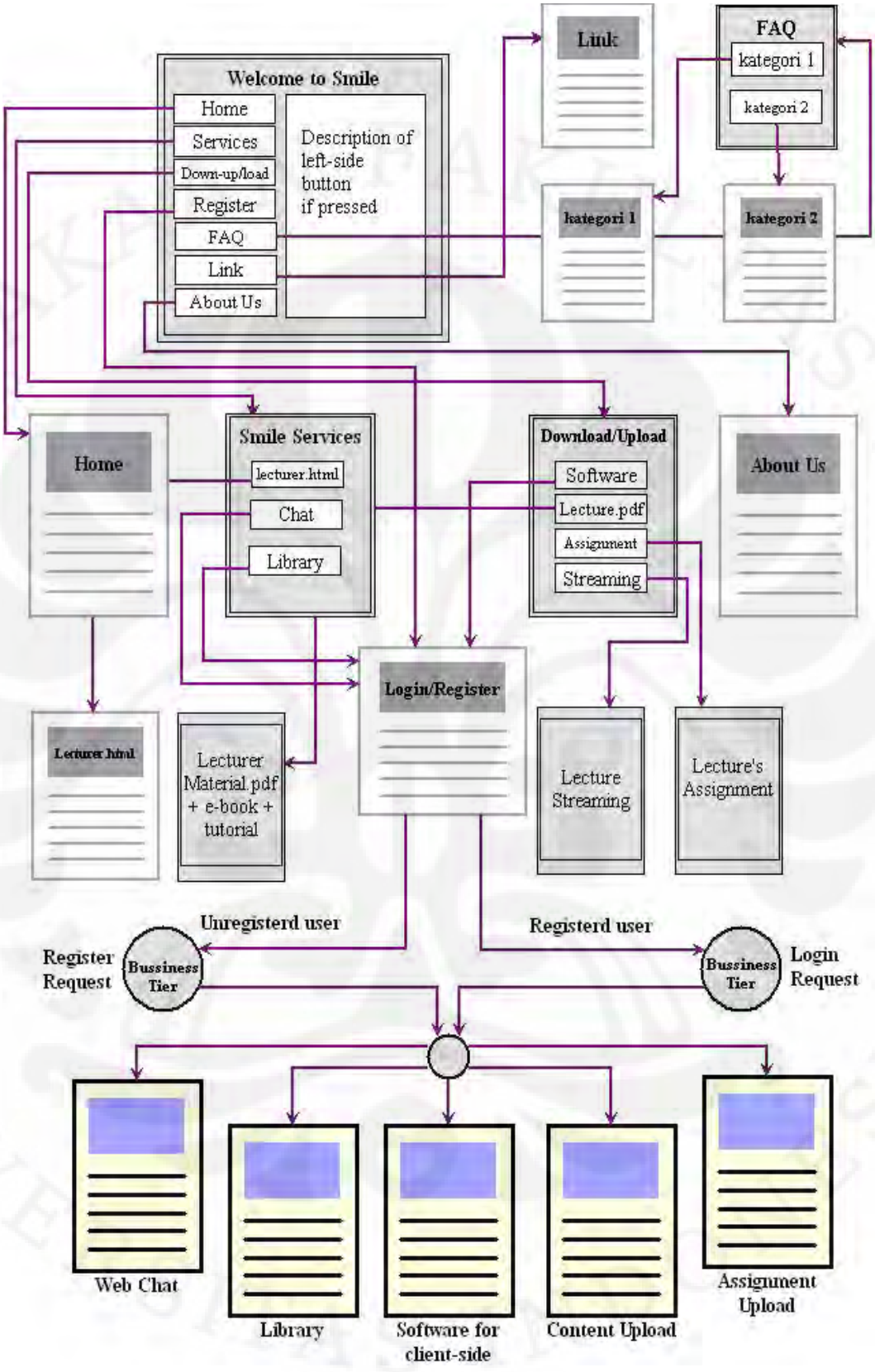
Implementasi komponen terdistribusi sangat sulit dilakukan. Hal ini karena pemrogram perlu mengatur bagaimana masing-masing komponen agar dapat saling berkomunikasi. Misalnya jika dua klien akan mengakses data melalui *enterprise bean* yang sama siapa yang akan didahulukan. Belum lagi pengaturan terhadap *bean* yang harus menjaga informasi kliennya hingga pada waktu tertentu. Tentu ini akan jauh lebih rumit lagi jika klien yang mengakses sangat banyak, sehingga membutuhkan pengaturan memori yang baik dan pengetahuan akan kemampuan *concurrent request* dari kontainer serta tentunya pengaturan masalah *transaction*. Namun demikian, tentu penulis tidak akan membahas permasalahan rumit tersebut secara keseluruhan dan hanya akan membuat model komponen terdistribusi untuk *server* SMiLE dan bagaimana komponen-komponen tersebut saling berkomunikasi. Permasalahan rumit tersebut diserahkan pada kemampuan

kontainer Java.

3.5.1 Komponen *Web* dan Komponen Bisnis

Web SMiLE dibuat berdasarkan *frame work OpenCms*. Sebagai editor, penulis menggunakan *Netbeans IDE 5.5*. Interaksi setiap halaman *web* terlihat pada Gambar 3.5.

Komponen *web* berinteraksi dengan komponen bisnis terjadi pada halaman ”Daftar Register SMiLE”, yaitu dalam menangani register baru, daftar register dan edit register. Sebagai contoh akan dibahas satu aksi, yaitu daftar register. Pembahasan tidak tidak dilakukan secara terpisah antara komponen *web* dan komponen bisnis agar penjelasannya lebih mudah. Kode keseluruhan bagian tersebut dapat dilihat pada Lampiran 5 dan halaman *web* bagian tersebut diperlihatkan oleh Gambar 3.6



Gambar 3.5 Diagram halaman *web* SMiLE

Gambar 3.6 Halaman Daftar Registrasi SMiLE

Ketika halaman *Daftar Register SMiLE* pada menu utama *web* SMiLE di-*klik* halaman tersebut mengirimkan *request* ke kelas *RegistrasiSmileController.java*. Kelas Java inilah yang menangani seluruh *request* yang dikirimkan oleh halaman *Daftar Registrasi SMiLE*.

```
<h:dataTable value='#{registrasiSmile.registrasiSmiles}' var='item' border="1"
        cellpadding="2" cellspacing="0">

    ...

</h:dataTable>
```

Gambar 3.7 *Request* halaman JSP pada kelas *RegistrasiSmileController.java*

Dimana **`registrasiSmile`** pada **`registrasiSmile.registrasiSmiles`** diatas (lihat Gambar 3.7) merupakan objek yang dibuat dari kelas *RegistrasiSmile.java*, sebuah kelas *entity bean*. Hal ini berarti komunikasi antar komponen Java dalam bentuk pengiriman objek.

Pada kelas *RegistrasiSmileController.java*, *request* dari halaman *Daftar Register* SMiLE diimplementasikan dalam bentuk *method set* dan *get*, kemudian dilanjutkan dengan melakukan *query* kepada kelas *RegistrasiSmile.java* melalui

EntityManager dengan seperti ditunjukkan oleh dalam Gambar 3.8.

```
public class RegistrasiSmileController {
  ...
  public DataModel getRegistrasiSmiles() {
        EntityManager em = getEntityManager();
        try{
              Query q = em.createQuery("select object(o) from RegistrasiSmile as o");
              q.setMaxResults(batchSize);
              q.setFirstResult(firstItem);
              model = new ListDataModel(q.getResultList());
              return model;
        } finally {
              em.close();
        }
        }
        ...
}
```

Gambar 3.8 Panggilan method *getRegistrasiSmiles()* di kelas *RegistrasiSmilesController.java* ke kelas *RegistrasiSmile.java*

Pada kelas *RegistrasiSmile.java* setiap data diambil, disimpan sebagai objek dan dapat diakses melalui *method set*, untuk menyimpan data ke dalam *database* dan *get*, untuk mengambil data dari *database* seperti ditunjukkan oleh Gambar 3.9.

```
@Entity
@Table(name = "REGISTRASI_SMILE")
@NamedQueries( {
      ...
      @NamedQuery(name = "RegistrasiSmile.findByNama", query = "SELECT r
      FROM RegistrasiSmile r WHERE r.nama = :nama"),
      ...
})
public class RegistrasiSmile implements Serializable {

  ...
  @Column(name = "NAMA")
  private String nama;
  ...
  public String getNama() {
     return this.nama;
  }
  public void setNama(String nama) {
     this.nama = nama;
  }
  ...
}
```

Gambar 3.9 Kelas *RegistrasiSmile.java* mengambil data dari *database*

Kemudian halaman JSP menampilkan hasilnya dalam Gambar 3.10

```
<h:dataTable value='#{registrasiSmile.registrasiSmiles}' var='item'
        border="1" cellpadding="2" cellspacing="0">

                ...
                <h:column>
                  <f:facet name="header">
                    <h:outputText value="Nama"/>
                  </f:facet>
                  <h:outputText value="#{item.nama}"/>
                </h:column>
                ...

</h:dataTable>
```

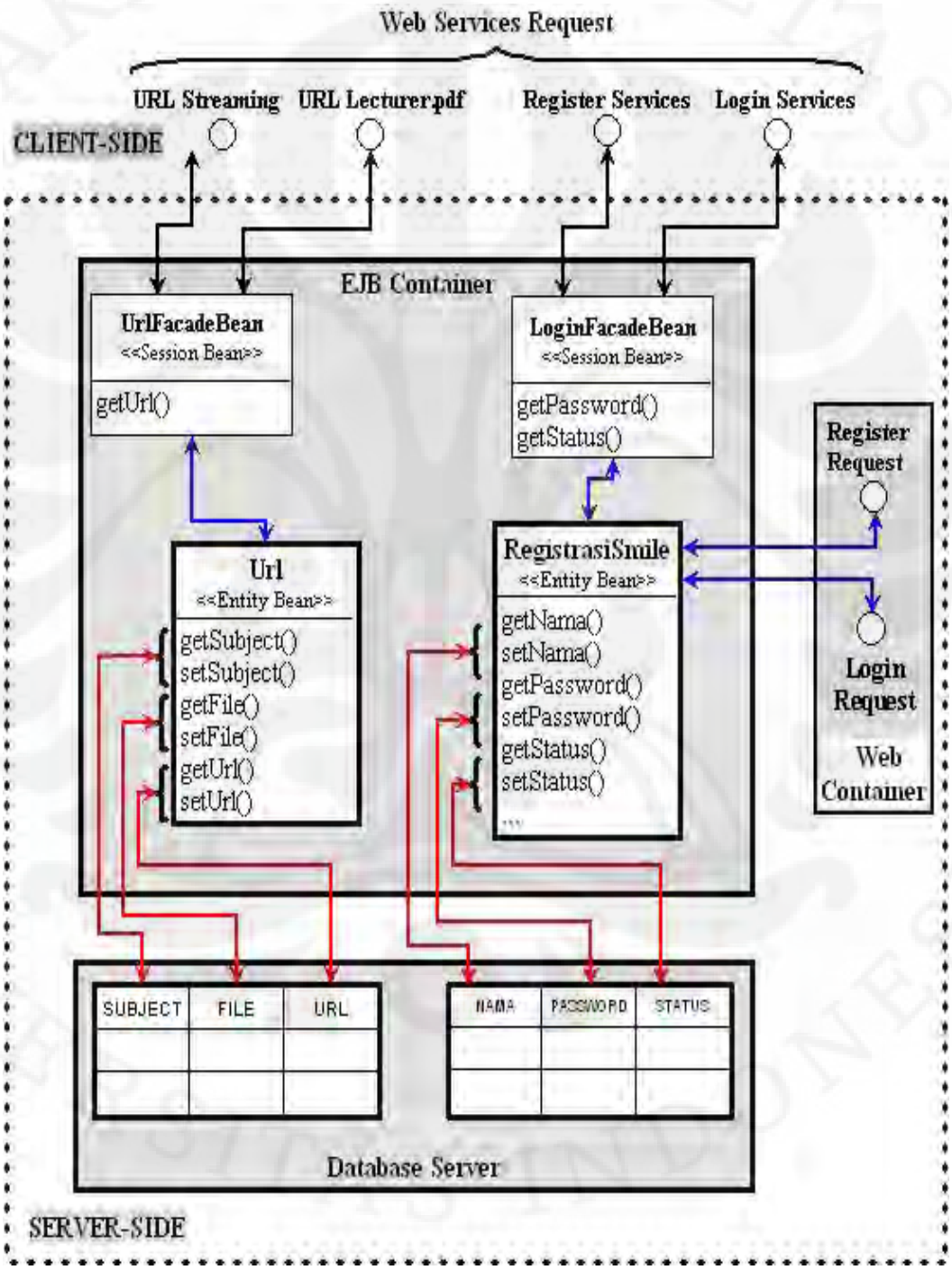
Gambar 3.10 Menampilkan hasil *request* pada halaman JSP

Penggunaan tiga komponen diatas, yaitu : halaman JSP, kelas Java *RegistrasiSmileController.java* dan *entity bean RegistrasiSmile.java* mengadopsi konsep *Model-View-Controler* (*MVC*). MVC merupakan konsep yang memisahkan komponen *user interface* yang mengatur tampilan (*View*), pengontrol yang menangani *request-response* dari *user interface* (*Controller*) dan komponen *persistence* yang menyimpan data secara tetap (*Model*). Berdasarkan konsep tersebut, halama *JSP* sebagai *View*, kelas Java *RegistrasiSmileController.java* sebagai *Controller* dan *RegistrasiSmile.java* sebagai *Model*. Penggunaan konsep ini sebenarnya lebih cocok digunakan pada sistem *multi-pages* dengan *multi-services* yang menuntut skalabilitas, reliabilitas dan sekuritas yang baik. Pada SMiLE, konsep ini menambah kerumitan tersendiri bagi penulis. Namun demikian MVC tetap penulis gunakan untuk memenuhi judul skripsi ini, yaitu *Perancangan dan Pengujian Arsitektur Komponen Multi-tier Terdistribusi untuk SMiLE Server Berbasis* Java EE.

3.5.2 Komponen Bisnis dan *Web Service*

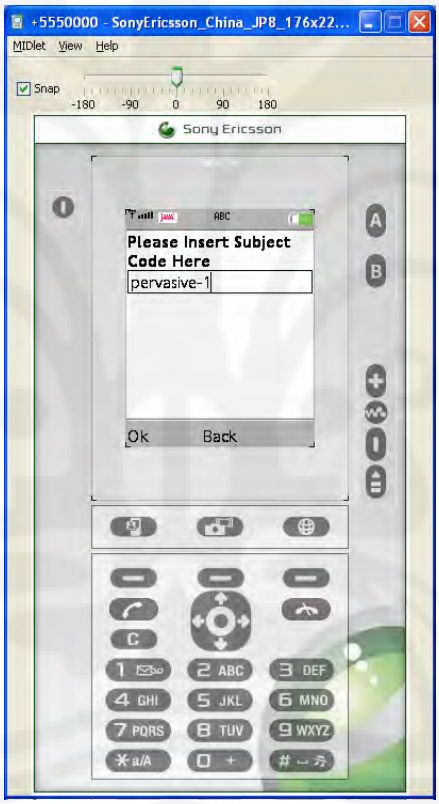
Pada bagian ini akan dibahas komunikasi antara klien *web service* pada *mobile client* dan *session bean* pada kontainer EJB. Layanan yang ditawarkan oleh *web service* adalah informasi *link* data tertentu yang berada pada *server* SMiLE dan proses konfirmasi agar klien mendapat hak akses tertentu. Kebutuhan akan kecepatan dalam mendapatkan pelayanan tentu menjadi tujuan utama. Oleh

karena itulah klien *web service* langsung berhubungan dengan *session bean*. Hal ini lebih efisien jika dibandingkan dengan klien *web service* harus menghubungi *servlet* pada kontainer *web* terlebih dahulu kemudian menuju *session bean*, yang juga akan memperberat kinerja *server* karena melakukan dua kali kerja. Implementasinya diperlihatkan oleh Gambar 3.12..



Gambar 3.11 Diagram kelas *enterprise bean* pada kontainer EJB

Jika seorang klien menggunakan aplikasi SMiLE pada *handset*-nya, misalnya untuk menjalankan aplikasi *streaming* seperti ditunjukkan oleh Gambar 3.11, Maka aplikasi klien akan mengirimkan *request* pada *stub generator* (*local client*). Lihat Sub-sub Bab 2.2.3 Interaksi Klien dengan *Web Service*. Pada *stub generator* inilah terdapat *method* yang dapat menangani *request* tersebut, sebagai referensi dari *server web services*. Kemudian *stub generator* mengirimkan pesan berbasis XML, SOAP, ke *server web service*. Pesan ini berisi parameter yang diminta oleh aplikasi klien, dalam hal ini meminta alamat URL. Pesan tersebut kemudian ditangani oleh kelas *UrlFacadeBean.java*. Lihat Gambar 3.13.



Gambar 3.12 *Request service* dari klien *Web service*

```
@Stateless
@WebService
public class UrlFacadeBean {

    @PersistenceContext
    private EntityManager em;

    /** Creates a new instance of UrlFacadeBean */
    public UrlFacadeBean() {
    }

    public String getUrl(String subject) {
        String url = "-1"; //No match
        Query query = em.createNamedQuery("Url.findBySubject");
        query.setParameter("subject", subject);
        List <Url> urls = query.getResultList();
        if (!urls.isEmpty()) {
            Url c = urls.get(0); // #Get the first matching customer
            url = c.getUrl().toString();
        }
        return url;
    }

}
```

Gambar 3.13 Kelas *UrlFacadeBean.jav*a

Pesan tersebut kemudian ditangani dengan membuat objek *query* dari kelas *javax.persistence.Query*, dan memanggil kelas *entity bean Url.java* dengan mencari URL yang cocok berdasarkan parameter *subject*. Tanda *@Stateless* menujukkan bahwa ini adalah kelas *stateless session bean* dan *@WebService* menunjukkan bahwa kelas ini menangani *request* dari klien yang mengetahui WSDL *web service* ini. Lihat Sub Bab 2.3 *Web Services*.

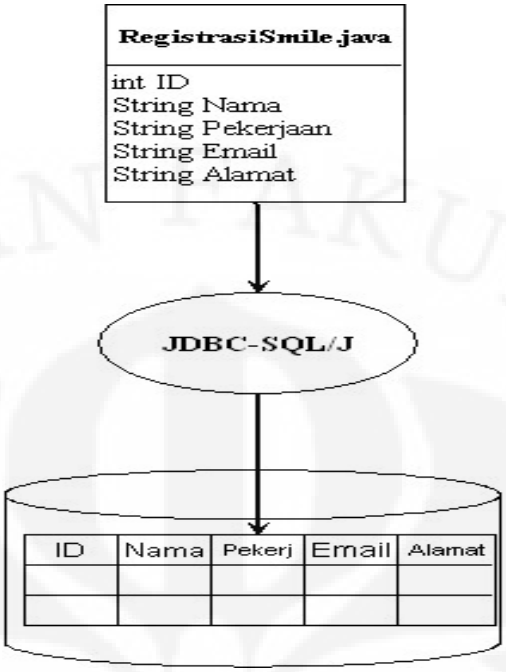
Pada kelas *Url.java*, berdasarkan parameter yang dikirim oleh *method getUrl()* dari kelas *UrlFacadeBean.java*, URL yang diinginkan diambil dan data tersebut dijadikan sebagai objek yang bisa diakses melalui *method* set dan *method get*. Lihat Gambar 3.14.

```
@Entity
@Table(name = "URL")
@NamedQueries( {
      @NamedQuery(name = "Url.findBySubject", query = "SELECT u FROM Url u
         WHERE u.subject = :subject"),
       ...
})
public class Url implements Serializable {

   @Column(name = "SUBJECT")
   private String subject;

   ...

   public String getSubject() {
      return this.subject;
   }

   public void setSubject(String subject) {
      this.subject = subject;
   }

   ...

}
```

Gambar 3.14 Kelas *Url.java* : Mengambil data dari *database*

3.5.3 Interaksi *Entity Bean* dan *Database*.

Interaksi antara *entity bean* (*Java Persistence* API) dengan *database* merupakan suatu proses yang sangat komplek. Ini disebabkan keterkaitan (*relational*) antara data yang satu dengan data yang lainnya. Contohnya ketika *user* melakukan registrasi melalui *Menu Utama* halaman *Daftar Registrasi SMiLE*. *User* memasukkan lima data yang saling berkaitan, yaitu : *ID, Nama, Pekerjaan, Email dan Alamat*. Data-data tersebut harus dijaga supaya tidak saling bertukar. Misal data ”Nama” *Bahrun Ulum* kapanpun diakses harus selalu mempunyai data ”Email” *milanelum@gmail.com*. Untuk menjaga data tersebut supaya tetap tidak saling tertukar ada suatu teknik yang dinamakan *object relational mapping*, yaitu membuat pemetaan terhadap data-data yang ada di dalam *database*. Pemetaan ini, di Java dilakukan oleh JDBC. Lihat Gambar 3.15.

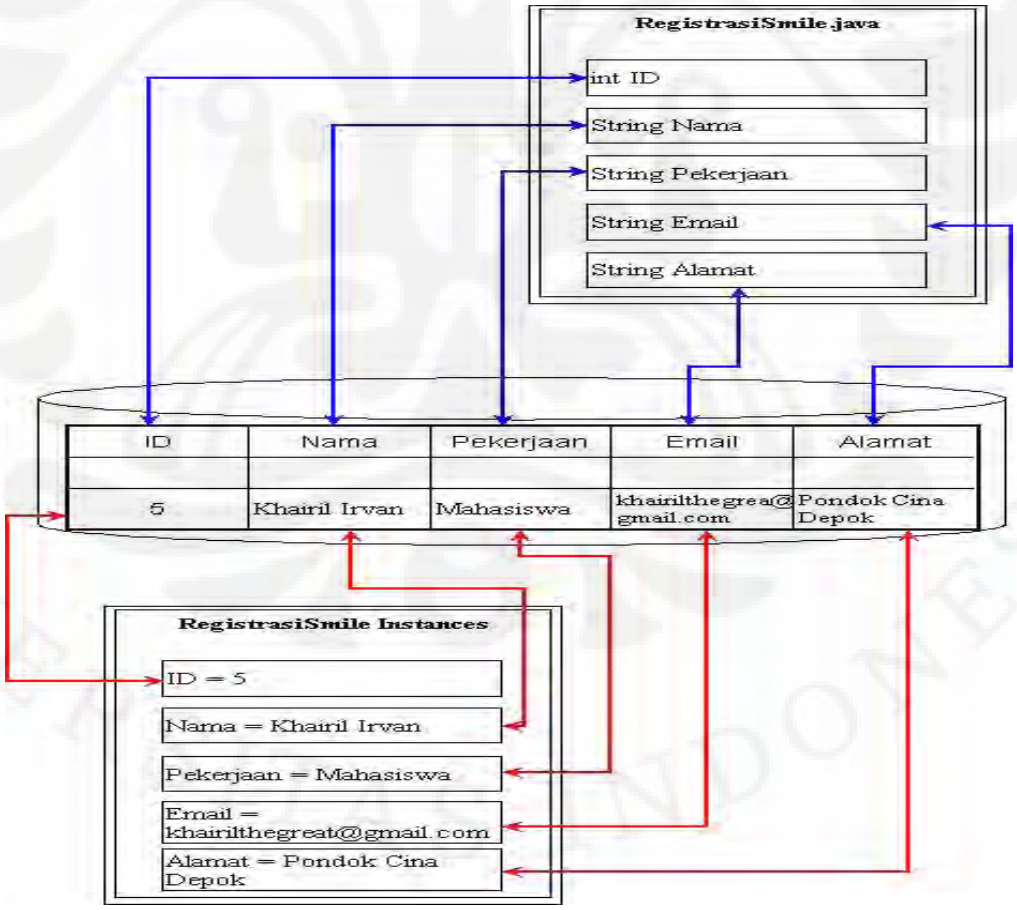


Gambar 3.15 Pemetaan yang dilakukan oleh JDBC

Misalnya seseorang akan melakukan registrasi dengan pada *web* SMiLE dengan data : ID=5, Nama=Khairil Irvan, Alamat=Pondok Cina, Depok, Pekerjaan=Mahaswa, Email=khairilthegreat@gmail.com. Lihat Gambar 3.16. Maka JDBC akan melakukan pemetaan sesuai dengan *field* pada *entity bean* seperti ditunjukkan oleh Gambar 3.17.

Pemetaan yang dilakukan oleh JDBC membuat data tidak bisa saling bertukar antara satu data dengan data lainnya. Setiap kolom data akan saling berkaitan (*relational*), sehingga jika didapat ID=5, maka nama user pasti adalah Khairil Irvan, begitu seterusnya. Keadaan ini tentu sangat menguntungkan untuk melakukan data *processing* yang lain, seperti : pencarian atau pengurutan.

Gambar 3.16 Registrasi pengguna SMiLE dengan data : ID=5, Nama=Khairil Irvan, Alamat=Pondok Cina, Depok, Pekerjaan=Mahaswa, Email=khairilthegreat@gmail.com



Gambar 3.17 Hasil pemetaan yang dilakukan oleh JDBC

BAB IV
UJI COBA DAN ANALISA SERVER SMILE

Sistem berbasis Java EE seperti SMiLE, menurut Floyd Marinescu (*www.theserverside.com*), memerlukan beberapa langkah untuk melakukan uji coba terhadap performa dan kemampaun skalabilitasnya, yaitu sebagai berikut :

4.1 UJI COBA *FUNCTIONAL*

Functional test merupakan uji coba yang umumnya dilakukan pertama kali terhadap banyak aplikasi. Uji coba ini dilakukan untuk mengetahui agar semua komponen berjalan dengan baik. *Functional test* dilakukan terhadap SMiLE dalam lingkungan *localhost* dengan cara mengakses tiap komponennya dengan menggunakan *web browser* (*Mozila Firefox*) dan *browser web service* dari *Sun Java System Application Server* dan *Netbeans IDE*.

4.2 UJI COBA *LOAD* DAN *SCALABILITY*

Load and scalabilty testing dapat dilakukan dengan dua cara berbeda, 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*). Pada skripsi ini hanya akan menguji coba SMiLE dengan metode kedua (*response time* karena peningkatan *concurrent users*) hal ini karena sampai saat ini (skripsi ini dicetak), penulis belum mendapatkan *tools* yang dibutuhkan untuk melakukan uji coba metode pertama (*response time* karena peningkatan *database*).

4.2.1 Persiapan Uji Coba

Tools yang digunakan untuk melakukan uji coba SMiLE :

1. *Personal computer* dengan spesifikasi : *Processor* AMD Sempron 2500, memori DDR SDRAM 1 GB, *hard disk* 40 GB, VGA dengan *chipset* Radeon 9550, Monitor 15”.

2. Apache JMeter 2.3.1

   Apache JMeter merupakan *software* gratis dan *open source* yang dibuat oleh *Apache Software Foundation* dan digunakan untuk menguji aplikasi berbasis *client-server*.

4.2.2 Tujuan Pengujian *Load* dan *Scalability*

Pengujian ini dilakukan untuk mengetahui kemampuan SMiLE dalam menangani peningkatan *request* dan pengaruh jumlah *user* terhadap kesalahan yang terjadi.

4.2.3 Proses Uji Coba

Uji coba menggunakan Apache JMeter dilakukan dengan memberikan HTTP *request* terhadap SMiLE. Uji coba dilakukan melalui tiga tahap, dimana masing-masing tahapan terdiri dari sembilan konfigurasi, seperti dijelaskan sebagai berikut (Lihat Gambar 4.1 sebagai contoh):

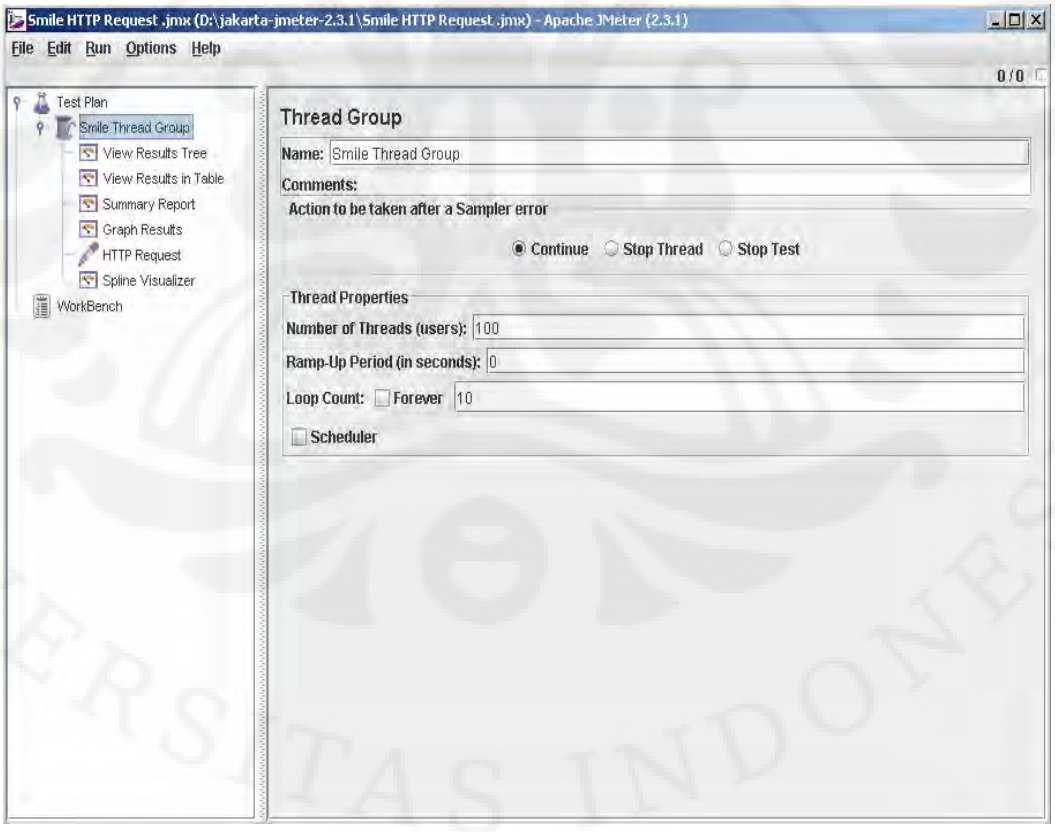
A. *Low load* dengan konfigurasi :
   1. *Low load* dengan lima *concurrent user*, *ramp up periode* nol dan *loop count* sepuluh (*Low load* 5-0-10).
   2. *Low load* 5-0-100.
   3. *Low load* 5-0-*forever*.
   4. *Low load* 5-10-10.
   5. *Low load* 5-10-100.
   6. *Low load* 5-10-*forever*.
   7. *Low load* 5-100-10.
   8. *Low load* 5-100-100.
   9. *Low load* 5-100-*forever*.

B. *Medium load* dengan konfiguarasi :
   1. *Medium load* 10-0-10.
   2. *Medium load* 10-0-100.
   3. *Medium load* 10-0-*forever*.
   4. *Medium load* 10-10-10.
   5. *Medium load* 10-10-100.

6. *Medium load* 10-10-*forever*.
7. *Medium load* 10-100-10.
8. *Medium load* 10-100-100.
9. *Medium load* 10-100-*forever*.

C. *High load* dengan konfiguarasi :

1. *High load* 100-0-10.
2. *High load* 100-0-100.
3. *High load* 100-0-*forever*.
4. *High load* 100-10-10.
5. *High load* 100-10-100.
6. *High load* 100-10-*forever*.
7. *High load* 100-100-10.
8. *High load* 100-100-100.
9. *High load* 100-100-*forever*.



Gambar 4.1 Uji coba terhadap SMiLE dengan konfigurasi *high load* 100-0-10.

Concurent user (*Number of Thread*) menunjukkan jumlah *user* yang mengakses SMiLE secara bersamaan. *Ramp-up periode* menunjukkan waktu yang dibutuhkan JMeter untuk membuat *thread*. *Ramp-up periode* nol berarti JMeter membuat *thread* secara bersamaan pada waktu nol detik pertama, *Ramp-up periode* sepuluh berarti JMeter membuat *thread* secara bersamaan pada waktu nol sampai sepuluh detik. Sedangkan *loop count* mununjukkan perulangan *request* yang diberikan kepada SMiLE, *forever* berarti *concurent request* yang diberikan oleh *concurent user* dilakukan secara terus-menerus dan jika *loop count* dalam bentuk angka berarti *concurent request* yang diberikan oleh *concurent user* dilakukan secara berulang sebanyak nilai *loop count* yang dimasukkan.

Concurent user pada pengujian ini tidak menunjukkan *concurent user* yang sebenarnya. Jika dibanding dengan *concurent user* ketika seorang mengakses *web*, menurut Floyd Marinescu, 10 *concurent user* pada pengujian menunjukkan 30-40 *concurent user* pada keadaan sebenarnya.

4.3 INTERPRETASI HASIL

Hasil uji coba menunjukkan kemampuan *response* SMiLE meningkat, seiring dengan jumlah *request* yang meningkat. Hal ini dapat dilihat pada Tabel 4.1. Pada konfigurasi *loop count forever*, penulis mengusahakan agar *sample request* yang diberikan kepada SMiLE mencapai lebih dari 5000. Ini dilakukan untuk mengetahui berapa banyak jumlah *sample* yang bisa direspon dengan baik. Data nomor 6 dari Tabel 4.1 diatas, *sample* yang di hasilkan tidak sampai 5000 disebabkan karena saat percobaan berlangsung komputer yang digunakan untuk percobaan juga digunakan untuk menjalankan aplikasi yang membutuhkan *resources processor* yang besar seperti *Netbeans IDE*, pemutar musik dan video. Begitu juga pada data nomor 26 tidak sampai 10000 (100 *concurrent user* dikali dengan 100 *loop count*).

Tabel 4.1 Hasil uji coba *Load* dan *Scalability*

No	Configuration	Concurent User	Sample	Throughput (response/minute)	Average response time (ms)
1	Low Load 5-0-10	5	50	4366	60
2	Low Load 5-0-100	5	500	4961	57
3	Low Load 5-0-forever	5	5081	4109	68
4	Low Load 5-10-10	5	50	368	9
5	Low Load 5-10-100	5	500	3281	11
6	Low Load 5-10-forever	5	4509	1299	166
7	Low Load 5-100-10	5	50	37	9
8	Low Load 5-100-100	5	500	369	8
9	Low Load 5-100-forever	5	5130	3563	39
10	Medium Load 10-0-10	10	100	3338	170
11	Medium Load 10-0-100	10	1000	5006	117
12	Medium Load 10-0-forever	10	5129	6154	93
13	Medium Load 10-10-10	10	100	657	13
14	Medium Load 10-10-100	10	1000	4449	47
15	Medium Load 10-10-forever	10	5070	4379	119
16	Medium Load 10-100-10	10	100	66	13
17	Medium Load 10-100-100	10	1000	656	9
18	Medium Load 10-100-forever	100	5060	4661	43
19	High Load 100-0-10	100	1000	7272	664
20	High Load 100-0-100	100	10000	5288	963
21	High Load 100-0-forever	100	5123	7815	752
22	High Load 100-10-10	100	1000	5393	160
23	High Load 100-10-100	100	10000	4933	951
24	High Load 100-10-forever	100	5160	6298	831
25	High Load 100-100-10	100	1000	604	8
26	High Load 100-100-100	100	9219	2650	345
27	High Load 100-100-forever	100	5112	2937	408

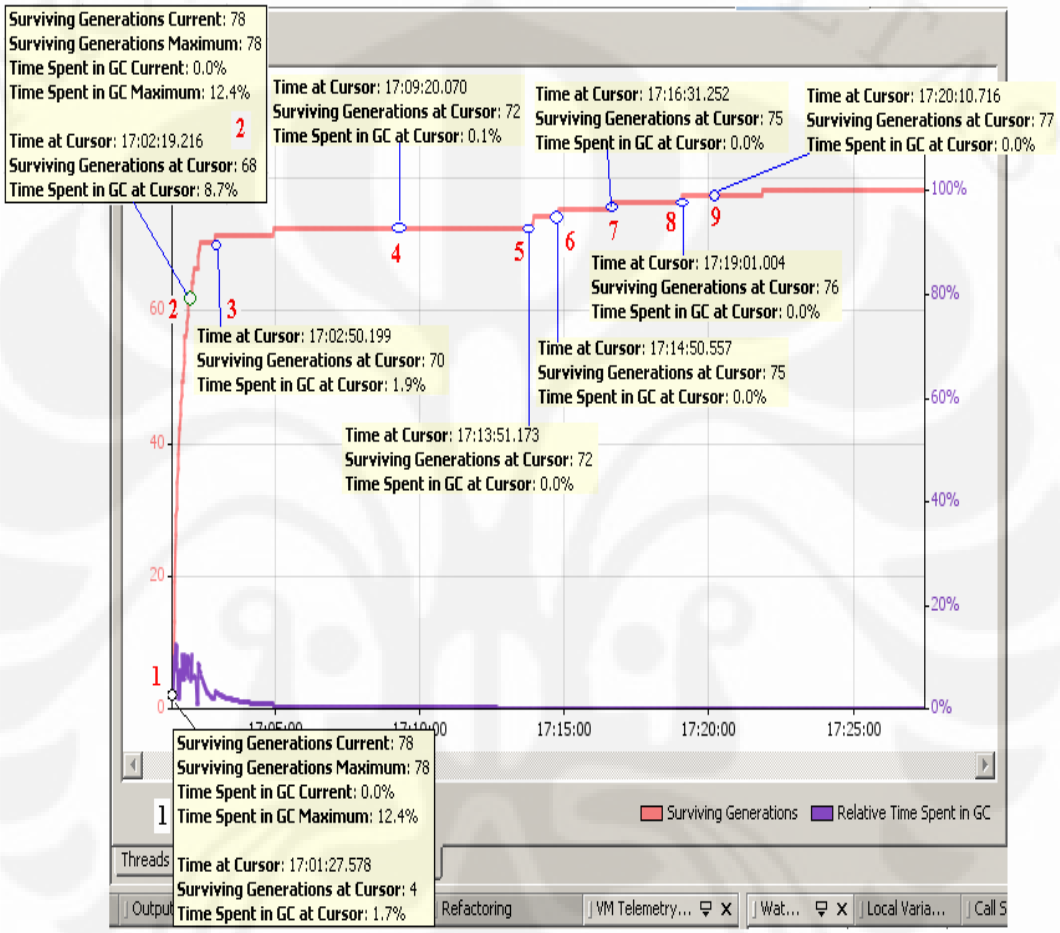
4.4 ANALISA

4.4.1 Analisa *Runtime Behavior*

Analisa ini dilakukan dengan menggunakan *tool* tambahan dari *Netbeans IDE* yaitu *Netbeans Profiler*. *Netbeans Profiler* adalah sebuah aplikasi yang digunakan untuk memonitor *runtime behavior* dari sebuah aplikasi yang sedang berjalan, seperti : penggunaan memori, *garbage collection*, jumlah *thread*, kondisi *thread* dan pembuatan objek.

4.4.1.1 *Garbage Collection*

Garbage collection adalah proses yang dilakukan untuk memperoleh kembali memori yang ditempati oleh objek yang tidak digunakan lagi sehingga memori tersebut bisa dimanfaatkan untuk menyimpan kembali aplikasi lain.



Gambar 4.2 Analisa *garbage collection*

- Keterangan Gambar 4.2 (Gambar aksi masing langkah berikut ada pada lampiran):
  1. *Build* SMiLE *Project*
  2. *Launching* SMiLE pada *browser*
  3. *Access* "Layanan SMiLE" pada menu utama
  4. *Download* file pdf
  5. Access "Daftar Registrasi SMiLE" pada menu utama
  6. *Click button* "Daftar"
  7. *Click button* "Create"
  8. *Click button* "Edit"
  9. *Click button* "Simpan"

- Sumbu-y menunjukkan jumlah memori dan presentase *relative time spent in GC*. Sedangkan sumbu-x, menunjukkan periode ketika SMiLE dijalankan.

Penggunaan *garbage collector* ini menjadi penting ketika SMiLE banyak mengakses *database*, karena setiap data dianggap sebagai objek dan sangat mungkin objek itu hanya untuk sekali pakai. Begitu juga ketika SMiLE diakses oleh banyak *user*.

Hasil percobaan bagian ini dapat dilihat pada Gambar 4.1. Grafik berwarna merah adalah *surviving generations* yang menunjukkan jumlah memori yang digunakan oleh berbagai objek dari berbagai waktu sejak digunakan oleh JVM selama percobaan bagian ini. Grafik berwana ungu adalah *relative time spent in GC*, menunjukkan waktu relatif yang diukur sebagai persentasi dari waktu yang dibutuhkan JVM dalam melakukan eksekusi *garbage collection* dibandingkan seluruh waktu dari *thread* yang tidak aktif (*suspend*).

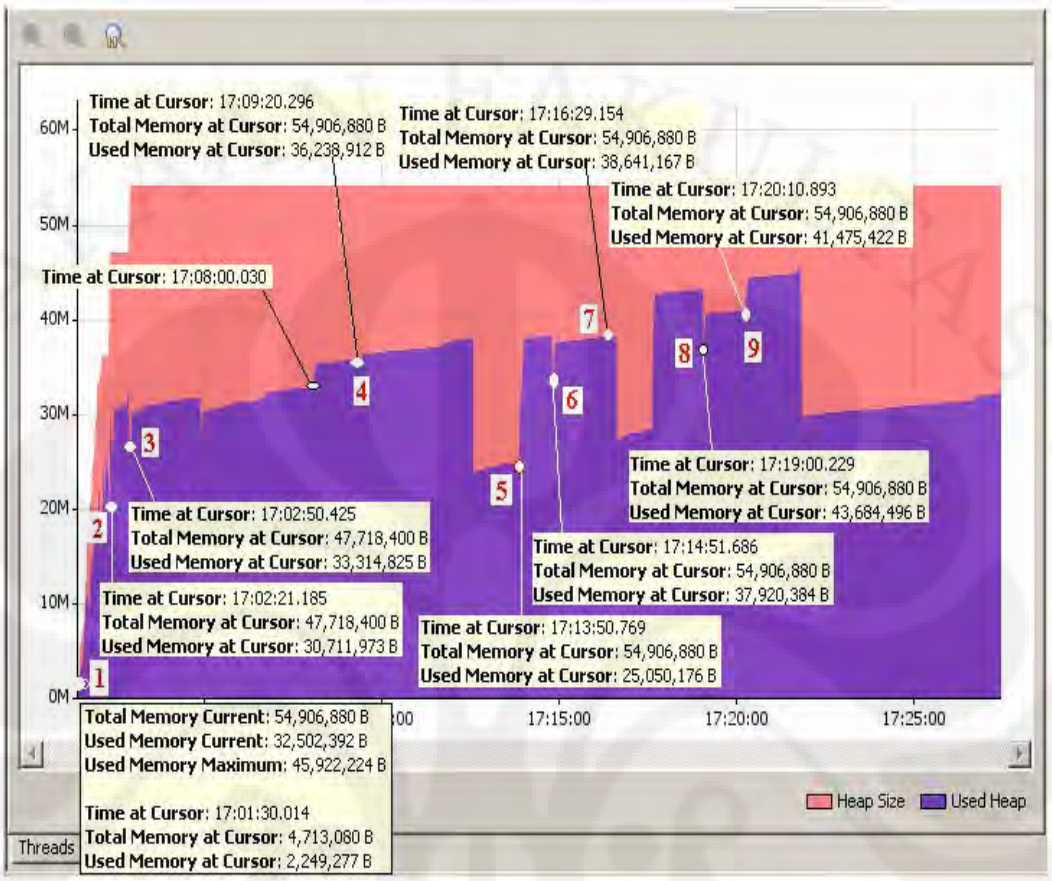
Pada awal grafik, yaitu ketika proses ”Build SMiLE Project”, pertambahan peningkatan grafik *surviving generations* sangat tajam begitu juga persentase *relative time spent in GC* mencapai angka yang terbesar 12.4 %. Kejadian ini dipengaruhi oleh JVM yang banyak meggunakan *instant* kelas Java untuk membangun SMiLE, sehingga banyak objek yang terbentuk. Karena objek itu hanya digunakan sekali (ketika proses *build*), maka JVM mengaktifkan *garbage collector* untuk menghapus objek itu sehingga waktu yang dibutuhkan untuk mengeksekusi *garbage collection* dibandingkan waktu dari jumlah objek yang tidak aktif mencapai nilai terbesar.

Pada waktu selajutnya *relative time spent in GC* menjadi lebih kecil, disebabkan objek-objek yang tercipta semakin sedikit ini dapat ditunjukkan oleh grafik *surviving generations* yang semakin landai. Peningkatan grafik *surviving generations*, kemudian, hanya dipengaruhi oleh jumlah klien yang mengakses SMiLE (lihat Gambar 4.2). Pada uji coba SMiLE, memori *leak* juga tidak terjadi. Memori *leak* terjadi jika grafik *relative time spent in GC* melewati grafik *surviving generations*.

4.4.1.2 Penggunaan Memori

Garbage collection menyebabkan penggunaan memori semakin mudah dikontrol. Hal ini terlihat dari Gambar 4.3, dimana memori yang digunakan tidak

pernah melewati total memori yang disediakan.



Gambar 4.3 Analisa penggunaan memori

- Keterangan Gambar 4.3 (Gambar masing-masing langkah berikut ada pada lampiran):
  1. *Build* SMiLE *Project*
  2. *Launching* SMiLE pada *browser*
  3. *Access* "Layanan SMiLE" pada menu utama
  4. *Download* file pdf
  5. *Access* "Daftar Registrasi SMiLE" pada menu utama
  6. *Click button* "Daftar"
  7. *Click button* "Create"
  8. *Click button* "Edit"
  9. *Click button* "Simpan"

- Sumbu-y menunjukkan jumlah memori. Sedangkan sumbu-x, menunjukkan periode waktu SMiLE dijalankan.

Adanya fungsi *garbage collection* pada Java sangat berguna untuk aplikasi *multi-tier* berskala besar. Gambar 4.3 menunjukkan bahwa Java akan melakukan *swap* terhadap memori pada periode waktu tertentu, hal ini ditunjukkan oleh pengurangan jumlah memori yang digunakan secara drastis.

Akses yang dilakukan oleh klien (Lihat keterangan Gambar 4.3) pada

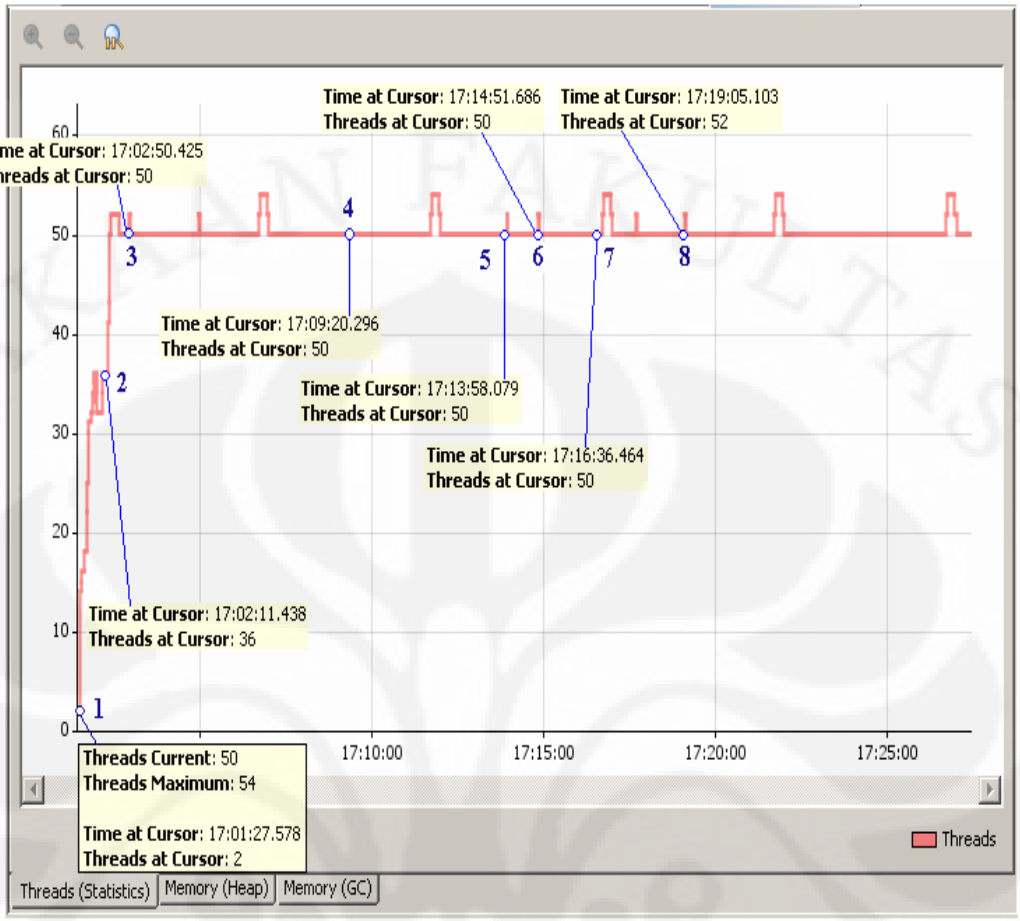
komponen *web* SMiLE tidak terlalu berpengaruh banyak, padahal penulis sudah menggunakan JSP sebagai komponen *web* (JSP dan *servlet* membutuhkan *runtime environment* untuk berjalan, sehingga akan mengunakan memori pada *server*. Lihat Sub-sub Bab 2.2.4 dan 2.2.5). Hal ini menunjukkan bahwa *servlet* disimpan dalam memori sejak proses *build* dan karena sistem tidak kekurangan memori, maka *servlet* tidak di-*swap*.

Penggunaan memori yang besar terjadi pada komponen bisnis. Setiap kali komponen bisnis diakses oleh klien (langkah 5-9), terjadi peningkatan jumlah memori yang digunakan cukup besar. Ini disebabkan karena setiap data yang diambil dari *database* akan dijadikan sebagai objek dan disimpan kedalam memori.

4.4.1.3 Multithread

Thread berarti eksekusi prosedur dalam sebuah program. Sedangkan *multithread* adalah kemampuan untuk melakukan eksekusi beberapa prosedur atau *method* dalam satu waktu tertentu. *Multithread* merupakan salah satu kemampuan yang penting dari Java. SMiLE yang dibangun dengan menggunakan Java tentu saja mempunyai kemampaun *multithread*. Kemampuan dari *multithread* SMiLE ditunjukkan oleh Gambar 4.4.

Thread yang dieksekusi meningkat pesat ketika SMiLE di-*build*. Setelah itu, rata-rata, jumlah *thread* yang dieksekusi selalu tetap. Perubahannya terjadi ketika SMiLE diakses oleh klien. Dari Gambar 4.4 terlihat terjadi peningkatan *thread* secara periodik sekaligus turun secara periodik. Sebelumnya sudah dijelaskan bahwa Java melakukan *swap* terhadap memori dengan melakukan *garbage collection*. Peningkatan *thread* itu disebabkan oleh aktivasi *garbage collector* JVM.



Gambar 4.4 Analisa *thread*

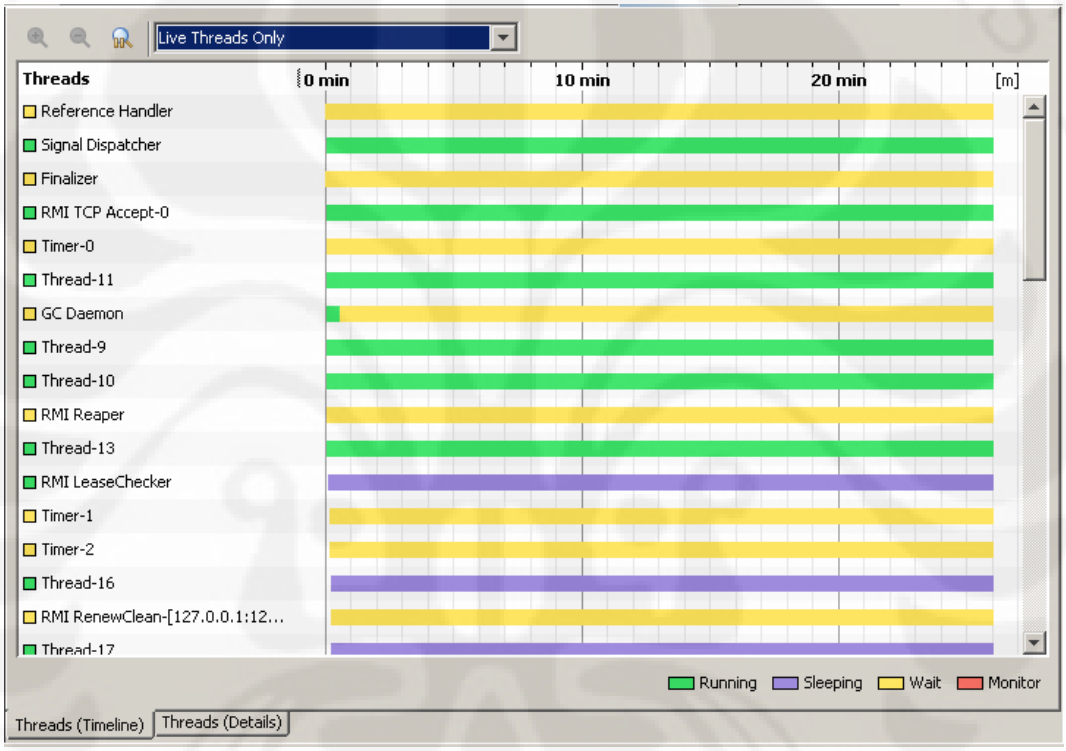
- Keterangan Gambar 4.4 (Gambar masing-masing langkah berikut ada pada lampiran):
  1. *Build* SMiLE *Project*
  2. *Launching* SMiLE pada *browser*
  3. *Access* "Layanan SMiLE" *pada menu utama*
  4. *Download* file pdf
  5. *Access* "Daftar Registrasi SMiLE" pada menu utama
  6. *Click button* "Daftar"
  7. *Click button* "Create"
  8. *Click button* "Edit"
  9. *Click button* "Simpan"

- Sumbu-y menunjukkan jumlah *thread* yang dieksekusi pada waktu bersamaan. Sedangkan sumbu-x, menunjukkan periode ketika SMiLE dijalankan.

Setelah waktu menunjukkan lebih dari 17:21:00, tidak ada aplikasi yang berjalan. Namun demikian tetap ada peningkatan dan penurunan *thread* secara periodik. Penyebabnya adalah *garbage collector* tetap diaktifkan meskipun tak ada aplikasi yang berjalan inilah yang menunjukkan *relative time spent in GC* pada Sub Bab 4.1, menjadi 0 %. Waktu eksekusi *garbage collection* adalah 0

detik, waktu *suspend thread* misal adalah t. Maka 0 detik dibagi t detik kali 100% adalah 0%.

Gambar 4.5 menunjukkan data historis *thread* dalam *timeline* diagram. Wana hijau menunjukkan *thread* siap dijalankan atau sedang berjalan. Warna kuning *thread* menunggu untuk dijalankan. Warna ungu *thread* berada dalam keadaan tidak aktif. Warna merah menunjukkan bahwa *thread* sedang di blok dan sedang melakukan sinkronisasi.



Gambar 4.5 Analisa *thread* (*by timeline1*)

Gambar 4.5 menunjukkan dengan jelas kondisi *thread* : *running*, *sleepng*, *wait* dan *monitor*.

4.4.2 Analisa Percobaan *Load* dan *Scalability*

Ketika *ramp-up periode* dinaikkan (Lihat Tabel 4.1), baik untuk konfigurasi *low* (Misal data ke-1, ke-4, ke-7), *medium* (Misal data ke-10, ke-13, ke-16), atau *high load* (Misal data ke-19, ke-22, ke-25), *throughput* (*response/minute*) menurun. Sedangkan pengaruh *ramp-up periode* terhadap *response time* berlaku berkebalikan.

Peingkatan jumlah *ramp-up periode* mengurangi kecepatan *sample request* yang dibuat oleh JMeter pada waktu tertentu. Misalkan uji coba dilakukan selama dua detik dengan lima *thread*. Jika *ramp-up periode* nol, maka seluruh *thread* akan dibuat secara bersamaan pada detik pertama percobaan dan langsung dikirimkan ke *server*. Misalnya pada detik berikutnya *response* sudah diberikan, maka *throughput* percobaan ini adalah 2.5 *response/second* (150 *response/minute*). Pada percobaan lain dengan waktu dan *thread* yang sama tetapi dengan *ramp-up periode* yang berbeda, misal lima detik. Jika pada detik pertama dibuat satu *thread* (Lima *ramp-up* untuk lima *thread*, berarti tiap detik dibuat satu *thread*) dan langsung dikirimkan, dan misalkan pada detik kedua diterima *response* serta dibuat *thread* kedua. Maka percobaan ini menghasilkan *throughput* 0.5 *response/second* (30 *response/minute*). Jadi jelas kenapa peningkatan *ramp-up* menurunkan *throughput*. Pengaruh *ramp-up periode* berlaku sebaliknya pada *response time*. Semakin banyak *thread* yang meminta *request* semakin memperberat kerja *server*. Seperti misalnya *entity bean*, bersifat *serial* dalam menangani *request* yang ditujukan kepadanya. Hal ini berarti sebuah *request* kemudian baru dapat diproses setelah *request* yang pertama selesai. Ini tentu akan memperbesar *average response time* secara keseluruhan. Berlawanannya nilai *ramp-up* dan *response time* ini akan menghasilkan nilai optimal, dimana nilai ini sangat bergantung pada kemampuan *hardware server* dan arsitektur *server* (SMiLE).

Pengaruh *loop count* dapat dilihat pada Tabel 4.1. Setiap tiga data secara berulang (Data 1-3, 4-6, 7-9 dan seterusnya), terjadi peningkatan *loop count* (Data 1-2, 4-5, 7-9 dan seterusnya) yang berarti terjadi peningkatan jumlah *thread* yang dihasilkan dan berarti juga peningkatan jumlah *sample/request*. Sedangkan peningkatan *concurrent user* terjadi pada tiap tahap percobaan, yang juga meningkatkan jumlah *sample*, yaitu data ke-1 dibandingkan dengan data ke-10 dan data ke-19, data ke-2 dibandingkan dengan data ke-11 dan data ke-20, begitu seterusnya (berulang setiap sembilan data).

Peningkatan jumlah *sample*, baik disebabkan karena bertambahnya *concurrent user* maupun karena bertambahnya nilai *loop count* dapat meningkatkan beban kerja *server* SMiLE, sehingga performa dan skalabilitas

SMiLE dapat diketahui. Dari Tabel 4.1 terlihat bahwa peningkatan beban juga meningkatkan *throughput*. Namun ketika *sample* mencapai angka sekitar 3800 atau SMiLE di uji coba dengan konfigurasi *high load* mulai terjadi *error*. Penyebabnya karena beban telah melebihi kemapuan kerja SMiLE untuk uji coba dengan spesifikasi komputer yang telah disebutkan sebelumnya (Sub-sub Bab 4.2.1). Ini dapat diketahui dari semakin lamanya *response time* dan waktu percobaan (waktu percobaan adalah jumlah *sample* dibagi *throughput*). Sedangkan data dengan konfigurasi *loop count forever* (Data ke-3 atau kelipatannya) menunjukkan kondisi ekstrem dari setiap peningkatan konfigurasi *ramp-up* untuk setiap tahap percobaan. Ini dilakakukan untuk mengetahui sejauh mana SMiLE mampu memberikan *response* dengan baik sebelum sample *error* terjadi.

BAB V
KESIMPULAN

1. Arsitektur komponen *multi-tier* sangat tepat digunakan untuk membuat sistem berskala besar yang mampu menangani peningkatan jumlah klien yang besar dengan banyak layanan yang ditawarkan.

2. Kemampuan *garbage colection* dari Java sangat berguna dalam hal efisiensi memori, terutama jika sistem diakses dalam jumlah besar. Hal ini dapat dilihat dari data percobaan, dari 54,9 *Megabyte* memori yang disediakan sistem, penggunaan memori maksimum hanya 45,9 *Megabyte* dan tidak pernah kekurangan memori selama sistem di uji coba.

3. *Garbage collector* bekerja sangat aktif pada awal sistem dijalankan, ini ditunjukan dengan *relative time spent in GC* mencapai 12.4 %.

4. *Garbage collector* akan tetap diaktifkan secara periodik setiap waktu 6-7 menit selama sistem dijalankan, ini terlihat dari grafik penggunaan memori yang menurun secara periodik dan peningkatan dan penurunan jumlah thread yang aktif juga secara periodik. .

5. Kemampuan menjalankan aplikasi *multithread* akan membuat proses komputasi berjalan lebih cepat, dalam SMiLE *thread* yang mampu berjalan secara bersamaan dapat mencapai 54 *thread*.

6. Kemampuan *concurent request* membuat sistem dapat diakses dalam jumlah besar dalam waktu yang bersamaan, sesuai dengan jumlah layanan yang ditawarkan.

7. *Web service* saat ini menjadi aplikasi antar muka yang paling tepat karena menawarkan portabilitas data, sehingga banyak jenis klien yang dapat mengaksesnya.

DAFTAR ACUAN

[1] Sari, Eunice Ratna. *Mobile Learning for Personal & Community Development in Developing Country*, http://engage-ist.org/fileadmin/engage/public_jakarta2006/presentations/A1_03_Sari_new.pdf. Diakses Tanggal 30-04-2007

[2] Deitel,H.M dan Deitel,P.J.*Java How to Program 6th Edition*, Prentice Hall Inc., NJ, USA, 2005, Hal 9.

[3] Jendrock,Erick,.dkk. The J2EE 1.4 tutorial, Sun Microsystem, Inc, SantaClara,California,U.S.A, 2006. Hal 10.

[4] Sun Microsystem,Inc., *What is Object*, http://java.sun.com/docs/books/tutorial/java/concepts/object.html. Diakses tanggal 7 November 2007.

[5] Jendrock,Erick,.dkk. *The Java EE 5 tutorial*, Pearson Education, Inc, Massachusetts,USA, February 2007, Hal 4.

[6] Jendrock,Erick,.dkk. *The J2EE 1.4 tutorial*, Sun Microsystem, Inc, SantaClara,California,U.S.A, 2006. Hal 7.

[7] Jendrock,Erick,.dkk. *The J2EE 1.4 tutorial*, Sun Microsystem, Inc, SantaClara,California,U.S.A, 2006. Hal 10.

[8] Kurniawan, Budi. *Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions*, New Riders Publishing, Indianapolis,USA, April 2002, Bab 2.

[9] Deitel,H.M dan Deitel,P.J.*Java How to Program 6th Edition*, Prentice Hall Inc., NJ, USA, 2005, Hal 1282.

[10] Jendrock,Erick,.dkk. *The Java EE 5 tutorial*, Pearson Education, Inc, Massachusetts,USA, February 2007, Hal 720

[11] Jendrock,Erick,.dkk. *The Java EE 5 tutorial*, Pearson Education, Inc, Massachusetts,USA, February 2007, Hal 719

[12] Sriganesh, Rima Patel., Brose, Gerald., Silverman, Micah. *Mastering Enterprise JavaBean™ 3.0*, Wiley Publishing, Inc,Indianapolis,USA,2006,Hal 59.

[13] Sriganesh, Rima Patel., Brose, Gerald., Silverman, Micah. *Mastering Enterprise JavaBean™ 3.0*, Wiley Publishing, Inc,Indianapolis,USA,2006,Hal 61.

[14] Bailey,Thomas., “Getting started with Web Services, JSR 172”, Sony Ericsson Mobile Communications AB, Lund, Sweden, September 2006. http://www.sonyericsson.com/developer. Diakses Tanggal 13-08-2007.

[15] Bray, Tim., Paoli, Jean., Sperberg-McQueen, C. M., Maler, Eve., “Extensible Markup Language (XML) 1.0 (2th Edition)”, *http://www.w3.org/*.

[16] Ortiz, C. E., “Introduction to J2ME Web Services”, Sun Microsystem Inc., http://developers.sun.com/mobility/reference/techart/index.html. Diakses tanggal 2-11-2007

[17] Sun Microsystem, Inc., “Java™ 2 Platform, Micro Edition (J2ME™) Web Services”, , A Technical White Paper, Juli 2004.

[18] Deitel,H.M dan Deitel,P.J.Java How to Program 6th Edition, Prentice Hall Inc., NJ, USA, 2005, Hal 9.

[19] Streaming in Mobile Networks. Paper dari MediaLab TeliaSonera Finland, Agustus 2004. http://www.medialab.sonera.fi/workspace/StreaminginMobileNetworksWP.pdf. Diakses Tanggal 07-06-2007.

[20] Yudono, Dianing Galih. ”Mobile Middleware Platform Berbasis Java untuk Ekstensi Aplikasi Enterprise”. ITB Indonesia. 2005.

[21] Keogh, James.*J2ME: The Complete Reference*, McGraw-Hill, Berkeley,California,2003.

DAFTAR PUSTAKA

Black, Jason T dan Hawkes, Lois Wright. *A Prototype Interface for Collaborative Mobile Learning.*

Brehovsky, Martin dkk. *Netbeans Mobility . NetBeans Software Day at 2005 JavaOne Conference*. 2005.

Georgiev, Tsvetozar., Georgieva, Evgenia., Smrikarov, Angel. *M-Learning - a New Stage of M-Learning*. http://ecet.ecs.ru.acad.bg/cst04/Docs/sIV/428.pdf, 2004.

McLean, Neil. *The M-Learning Paradigm: an Overview*. A Report for the Royal Academy of Engineeringand the Vodafone Group Foundation, Macquarie University, Sydney, November 2003.

Meleisea, Ellie (ed). *Workshop Report: Mobile Learning for Expanding Educational Opportunities*. Tokyo, Japan, 16 -20 May 2005.

Moore, Keith. Integrating Software Components, Hewlett-Packard Co., WICS'97, 28-29 July 1997.

Naismith, Laura., Lonsdale, Peter., Vavoula, Giasemi., Sharples, Mike. *Literature Review in Mobile Technologies and Learning*. University of Birmingham. http://www.futurelab.org.uk/research/reviews/reviews_11_and12/11_01.htm, 2006.

Niall Winters, *Mikes Sharples(ed) Big Issues in Mobile Learning,What is mobile learning*

O'Malley, C., Vavoula, G., Glew, J., Taylor, J., Sharples, M. & Lefrere, P. *Guidelines for learning/teaching/tutoring in a mobile environment. Mobilearn project deliverable*.

http://www.mobilearn.org/download/results/guidelines.pdf, 2003.

Pranata, Antony. *Development of Network Service Infrastructure For TranscodingMultimedia Streams*. Thesis Master di Fakultas Ilmu Komuter Universitas Stuttgart, Jerman, 2002. http://www.ilmukomputer.com.

Prasad, K.V. *Principles of Digital Communication Systems and Computer Networks*. Dreamtech Press, Massachusetts,USA, 2003.

Sari, Eunice Ratna. *Mobile Learning for Personal & Community Development in Developing Country*.

Sun Microsystem, Inc. *Netbeans IDE Field Guide* . USA. 2005.

Tanenbaum, A. S. *Computer Networks 4th Edition*. Prentice Hall Inc., NJ, USA, 2003.

Tan-Hsu Tan,. Tsung-Yu Liu., "The MObile-Based Interactive Learning Environment (MOBILE) and A Case Study for Assisting Elementary School English Learning", Department of Electrical Engineering National Taipei University of Technology Taipei, 106, Taiwan, ROC.

Yuen, Steve C. Summer Lecture Trip to China, May 28 - June 10, 2004.

Cisco Mobile Exchange (CMX) Solution Guide, Chapter 2 Overview of GSM, GPRS, and UMTS.

COMCAR Project. COMCAR – Communication and Mobility by Cellular Advanced Radio. http://www.comcar.de/overview.pdf.

Java™ Speech API Programmer's Guide. Sun Microsystems, Inc., http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/Preface.html. (2004, 1997-1998).

http://wiki.netbeans.org/wiki/view/ACompleteSOAAppNetbeans5.5 . 11 Agustus 2007

http://www.java2s.com/Code/Java/J2ME/AMIDletthatdisplaystheDoggyanimation.htm . 3 September 2007

http://testwww.netbeans.org/kb/55/amsecurity.html . 11 Agustus 2007

http://testwww.netbeans.org/kb/55/demo-end2end.html . 13 Agustus 2007

http://testwww.netbeans.org/kb/50/derby-demo.html . 7 Agustus 2007

http://www.theserverside.com/resources/article.jsp?l=Hemphill . 31 Agustus 2007

http://www.java2s.com/Code/Java/J2ME/TextBoxSharedClipBoard.htm . 3 September 2007

http://www.onjava.com/pub/a/onjava/2001/10/17/mobilej2ee.html?page=1 . 7 Agustus 2007

http://www.onjava.com/pub/a/onjava/2001/10/17/mobilej2ee.html?page=2 . 7 Agustus 2007

http://www.netbeans.org/kb/55/websvc-jax-ws.html . 11 September 2007

http://www.netbeans.org/kb/55/vwp-intro.html . 13 Agustus 2007

http://www.netbeans.org/kb/55/vwp-textcompletion.html . 27 Agustus 2007

http://www.developer.com/java/ejb/article.php/10931_3526721_1 . 7 Agustus 2007

http://www.developer.com/java/ejb/article.php/10931_3526721_2 . 7 Agustus 2007

http://www.developer.com/java/ejb/article.php/10931_3526721_3 . 7 Agustus 2007

http://www.developer.com/java/ejb/article.php/10931_3526721_4 . 7 Agustus 2007

http://www.developer.com/java/ejb/article.php/10931_3526721_5 . 7 Agustus 2007

http://www.infowatch.com/threats?chapter=162971949&id=207784708.

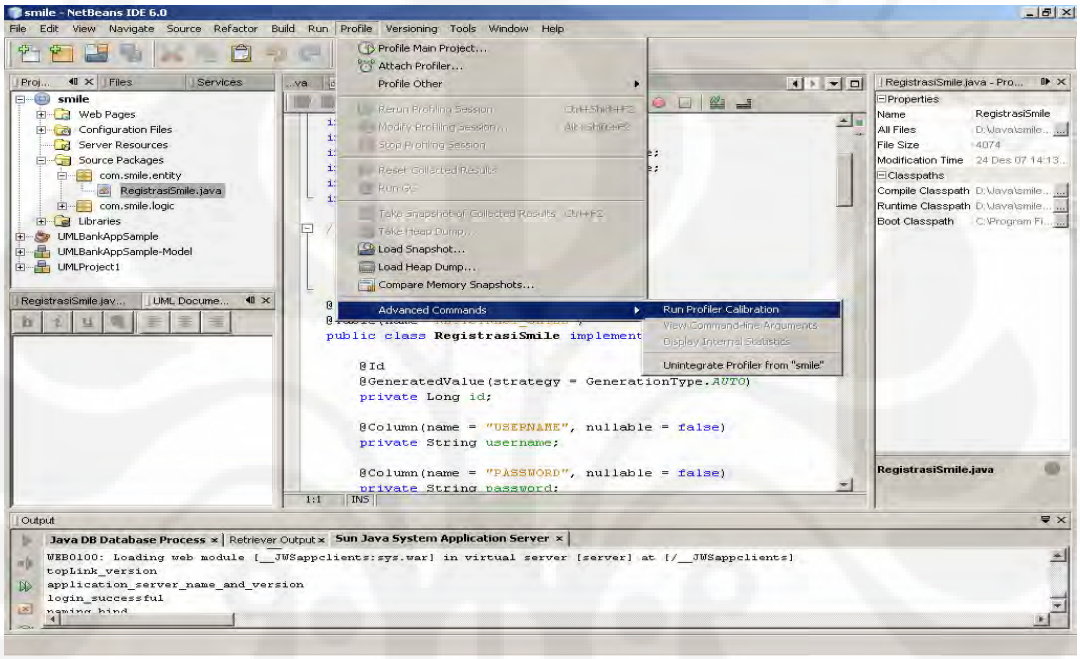
http://www.microsoft.com/windowsmobile/6/default.mspx.

http://en.wikipedia.org/wiki/Tablet_PC.

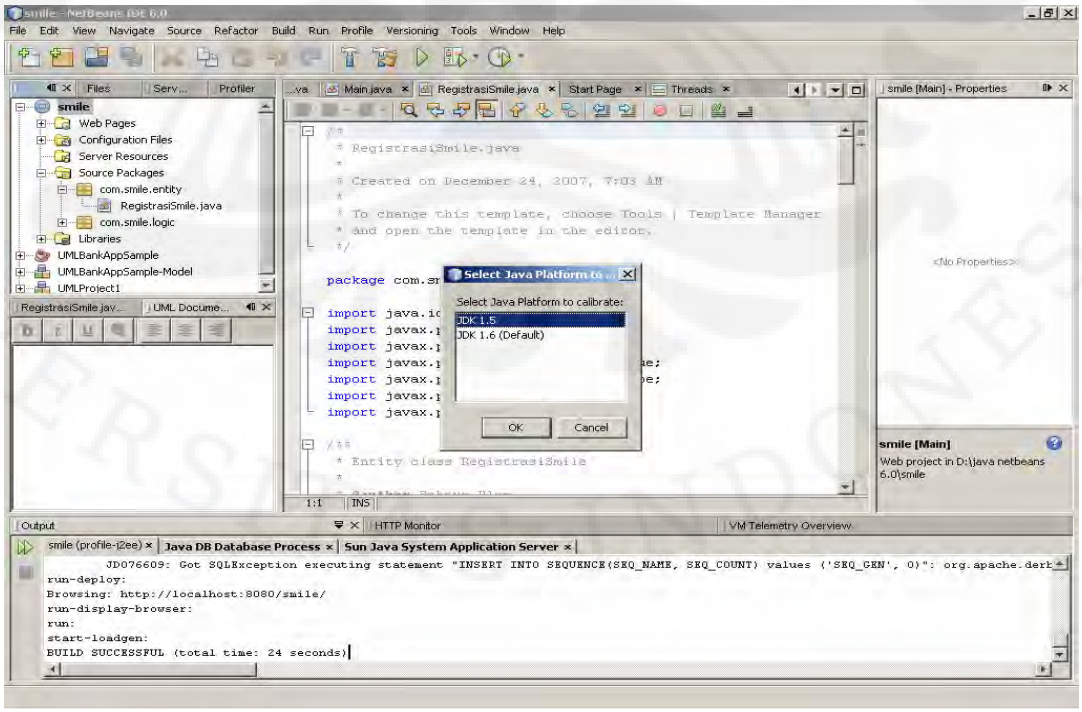
http://ilmukomputer.com/2006/09/11/mengenal-operating-system-untuk-mobile-devices/.

LAMPIRAN 1
LANGKAH MEMBUAT PROFILE SMILE

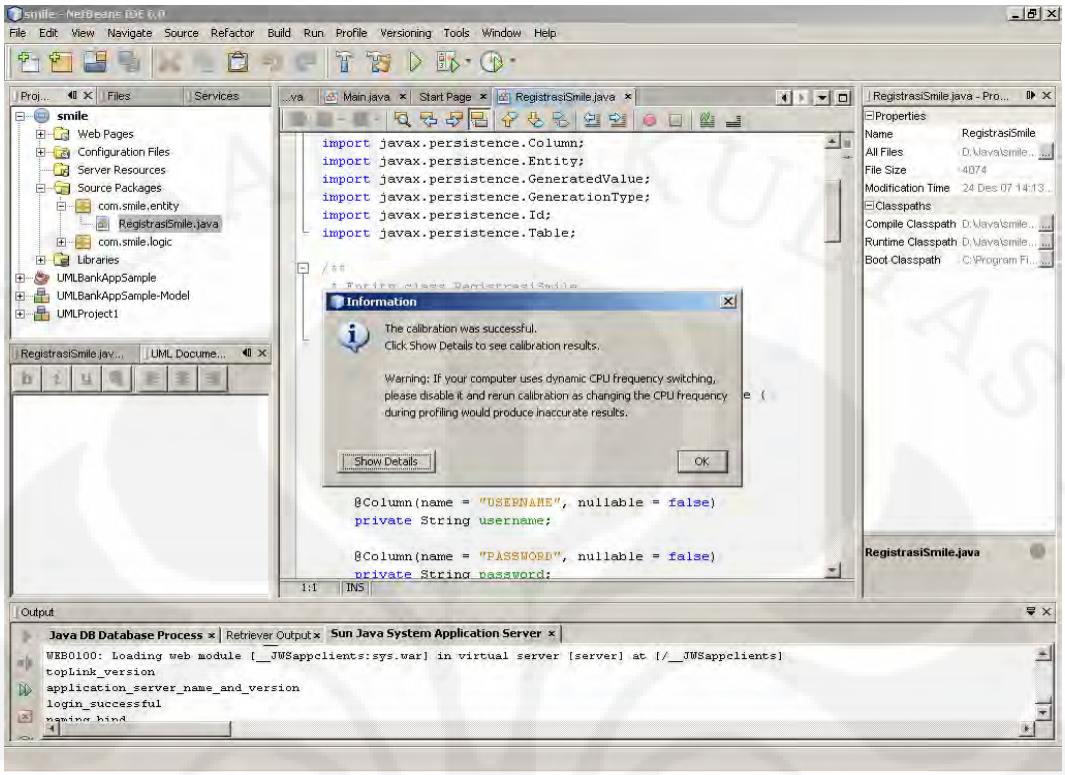
1. Pada menu utama pilih “Profile”> “Advanced Commands”> “Run Profiler Calibaration”



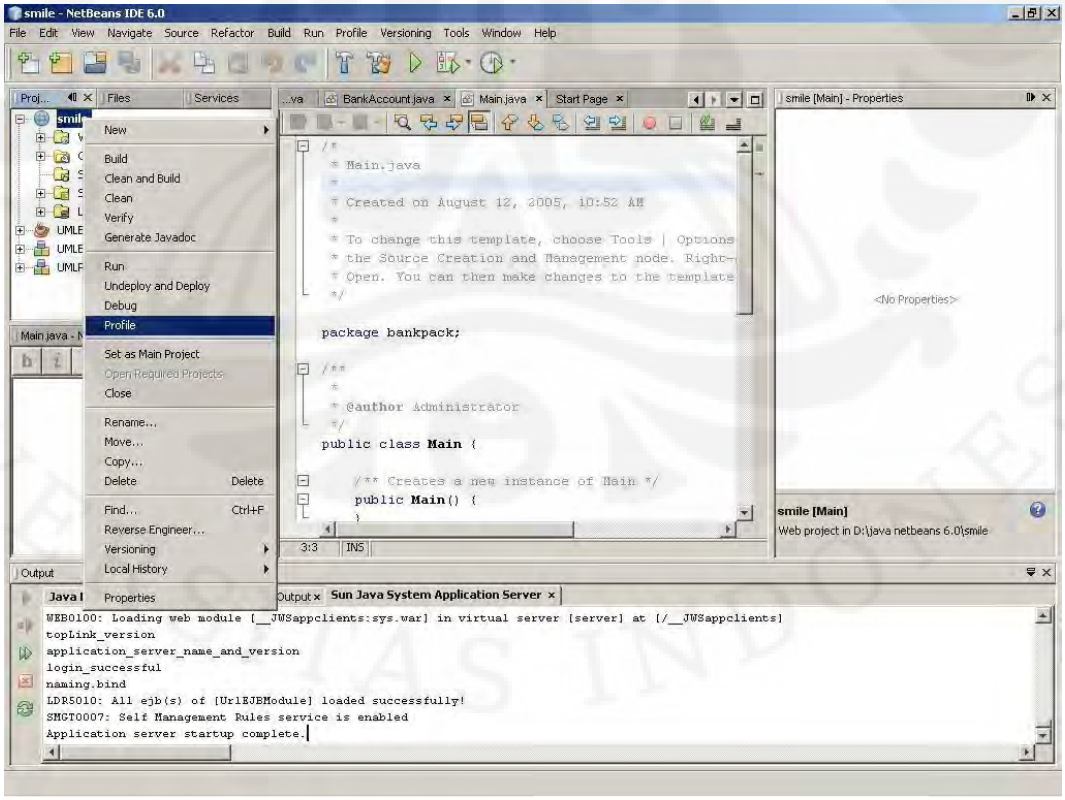
2. Pilih JDK untuk dikalibrasi agar bisa digunakan dalam melakukan *profile*> ”OK”



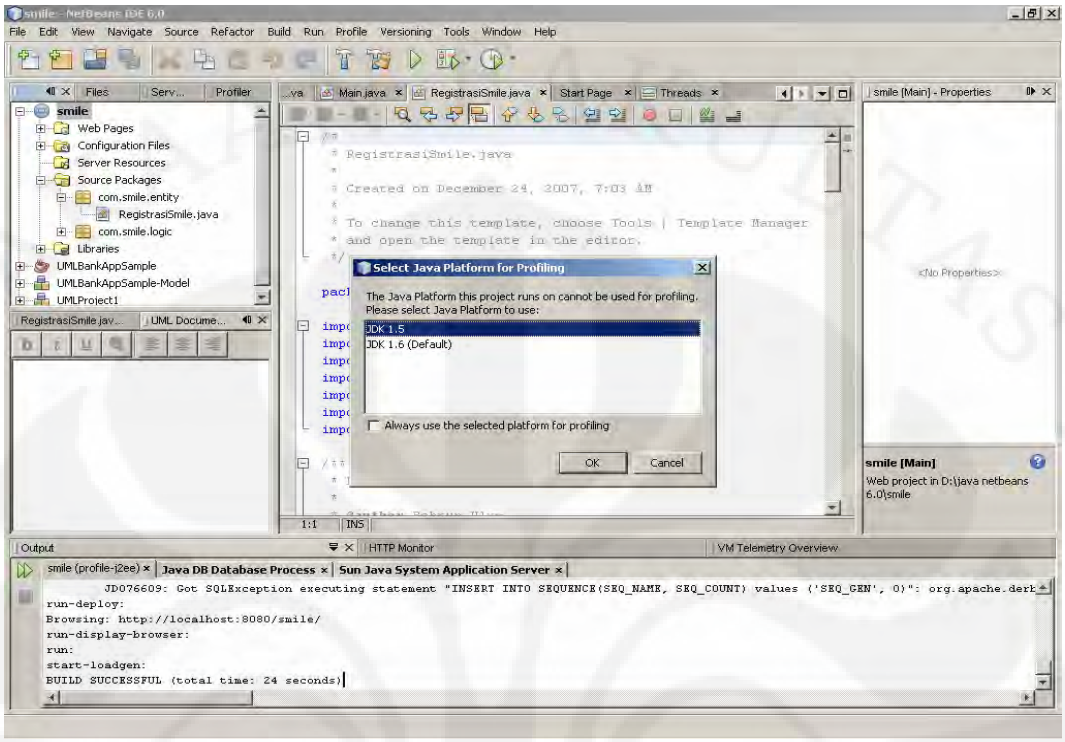
3. Muncul *window* “Information”. Jika *succesful* klik “OK”



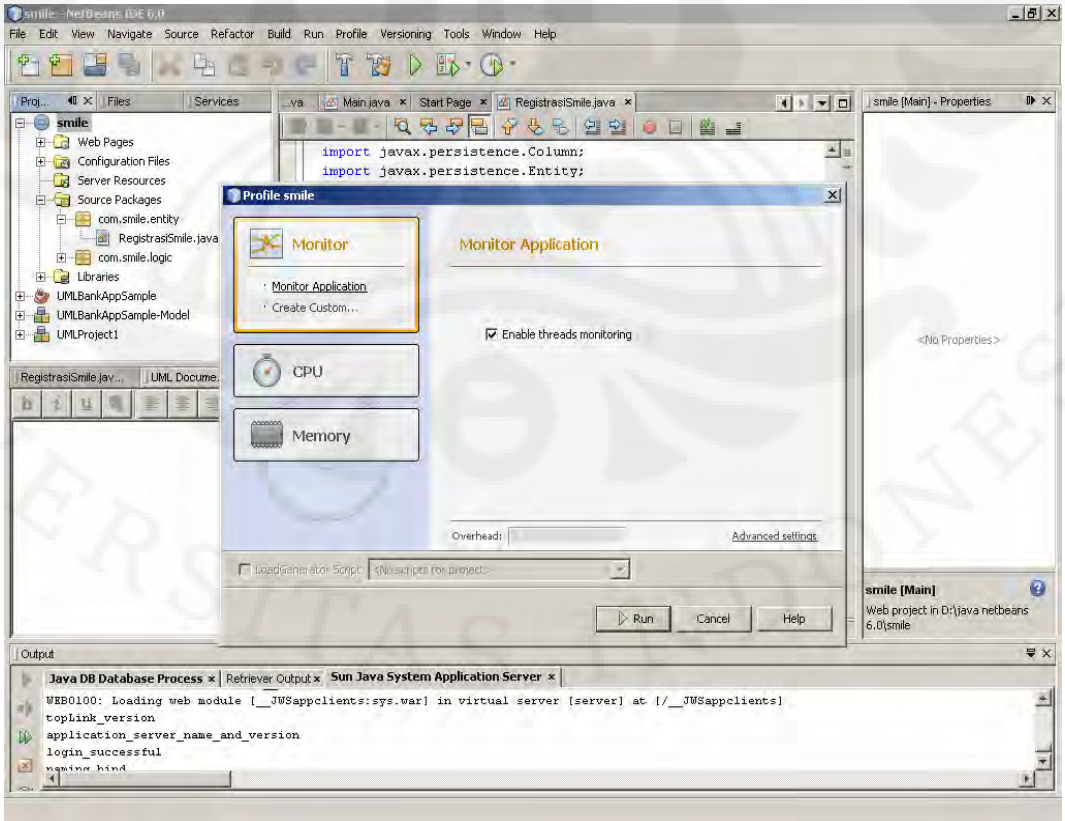
4. Klik kanan *project* SMiLE. Pilih “Profile” atau menu utama pilih “Profile”> “Profile Main Project”



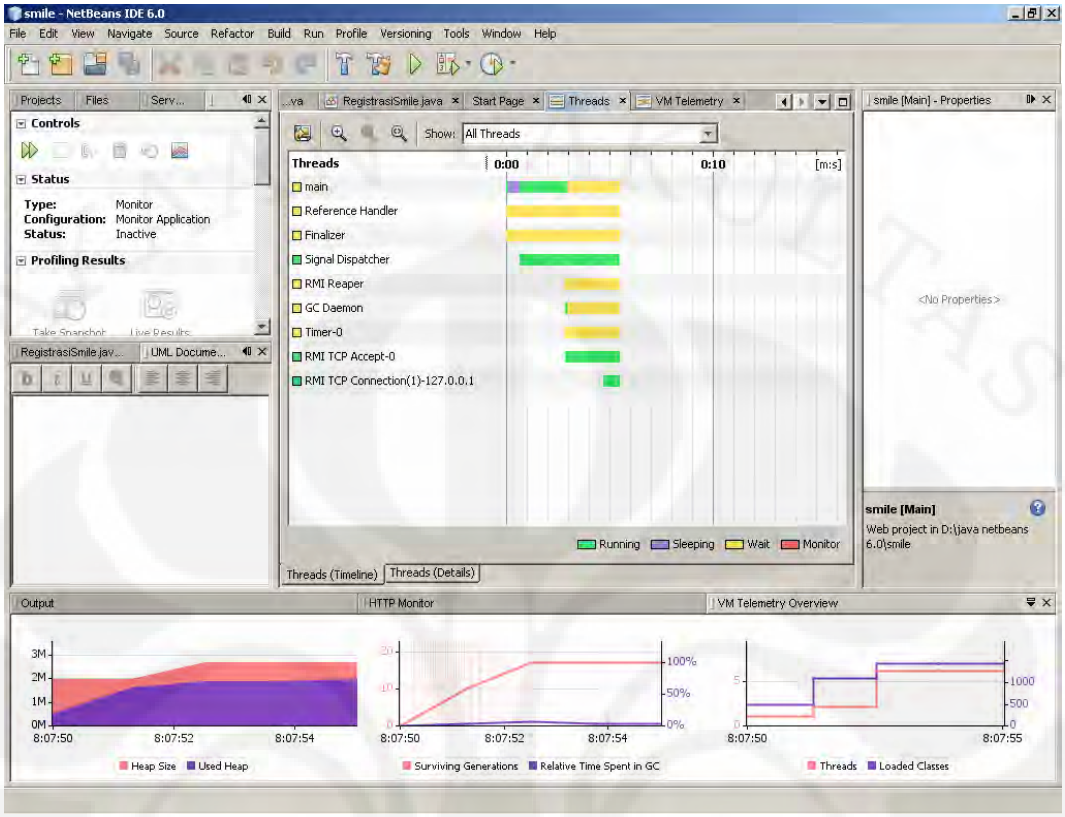
5. Kemudian pilih JDK (sudah dikalibrasi) yang ingin digunakan dalam melakukan *profile*



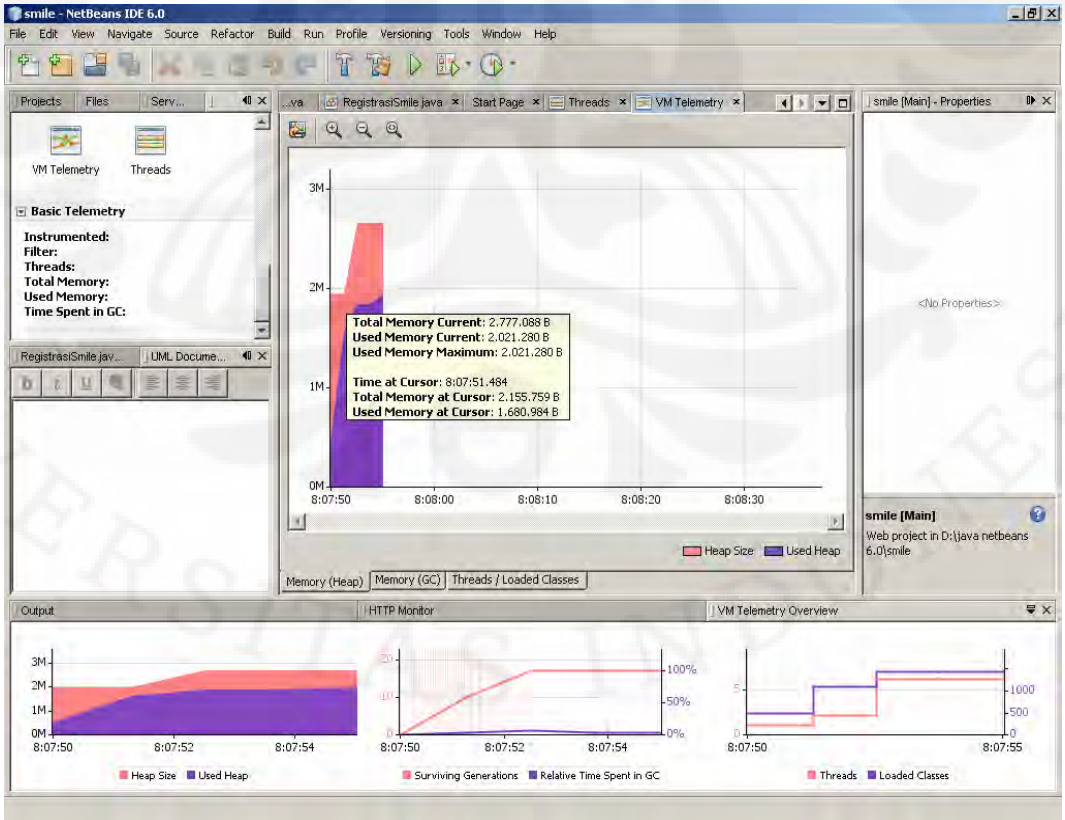
6. Setelah muncul *window* "Profile Smile", pilih aksi yang ingin dilakukan: "Monitor", "CPU", "Memory". Kemudian Klik "Run"



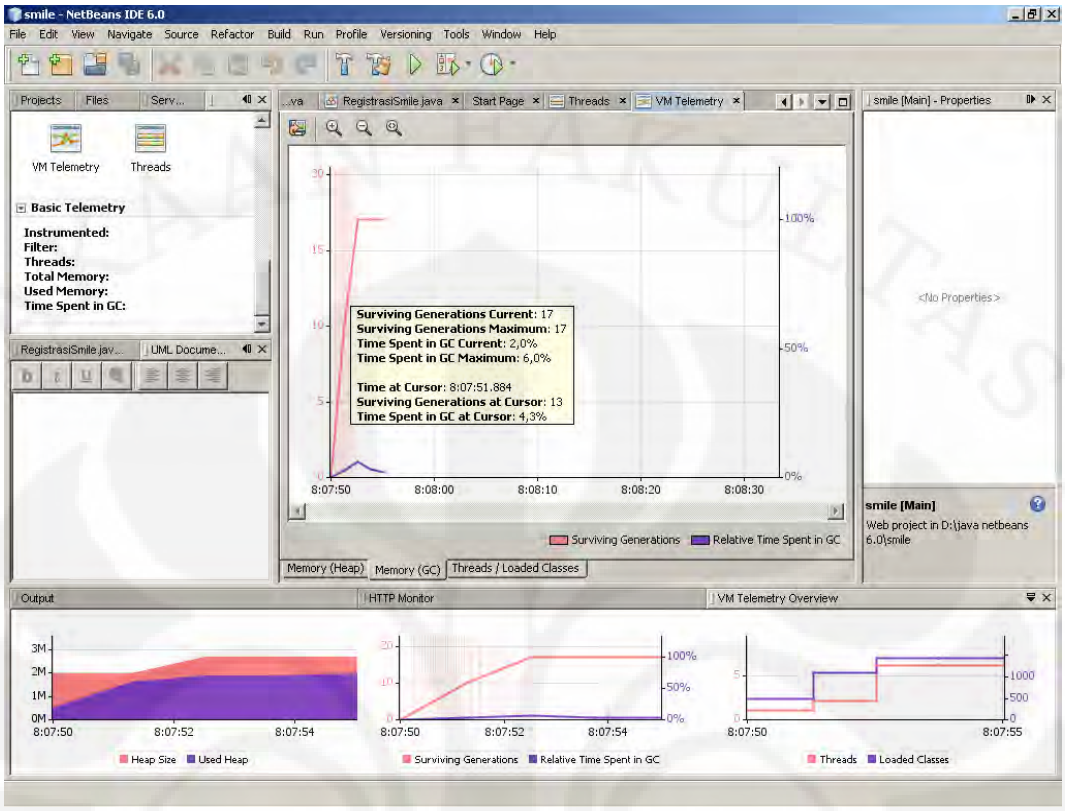
7. Proses “profile monitor” (1. *Monitoring thread in timeline*)



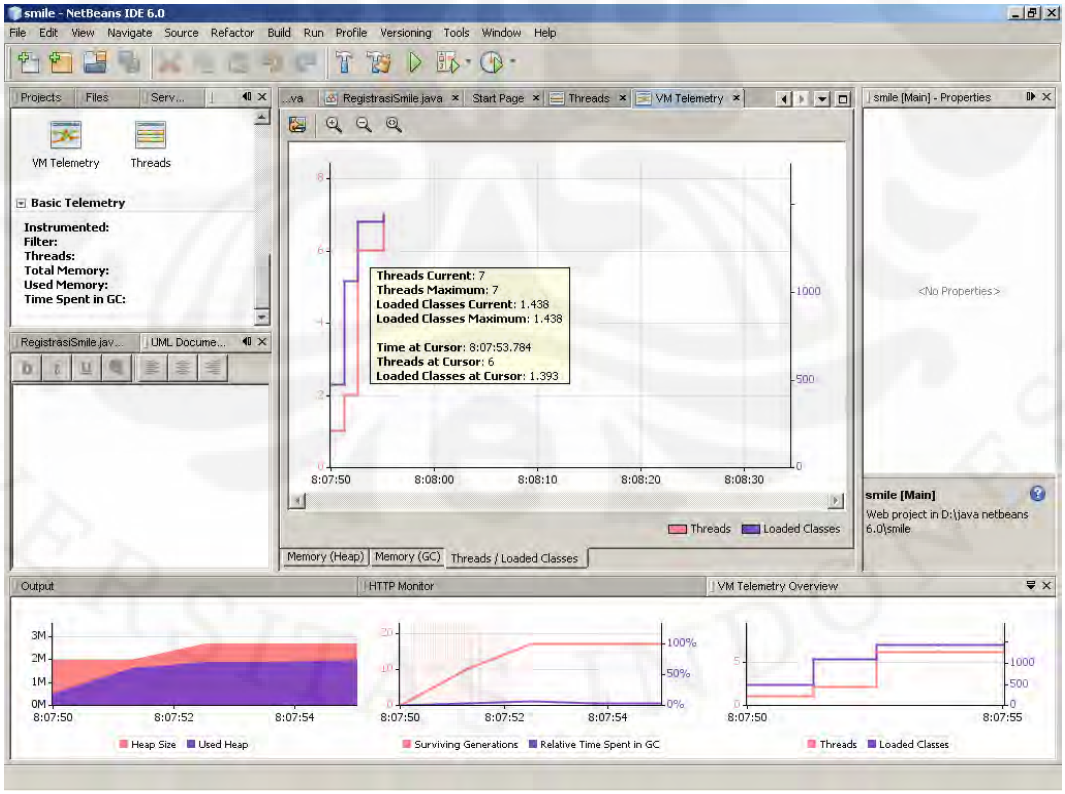
8. Proses “profile monitor” (2. *Monitoring heap size*)



9. Proses “profile monitor” (3. *Monitoring Garbage Collector*)

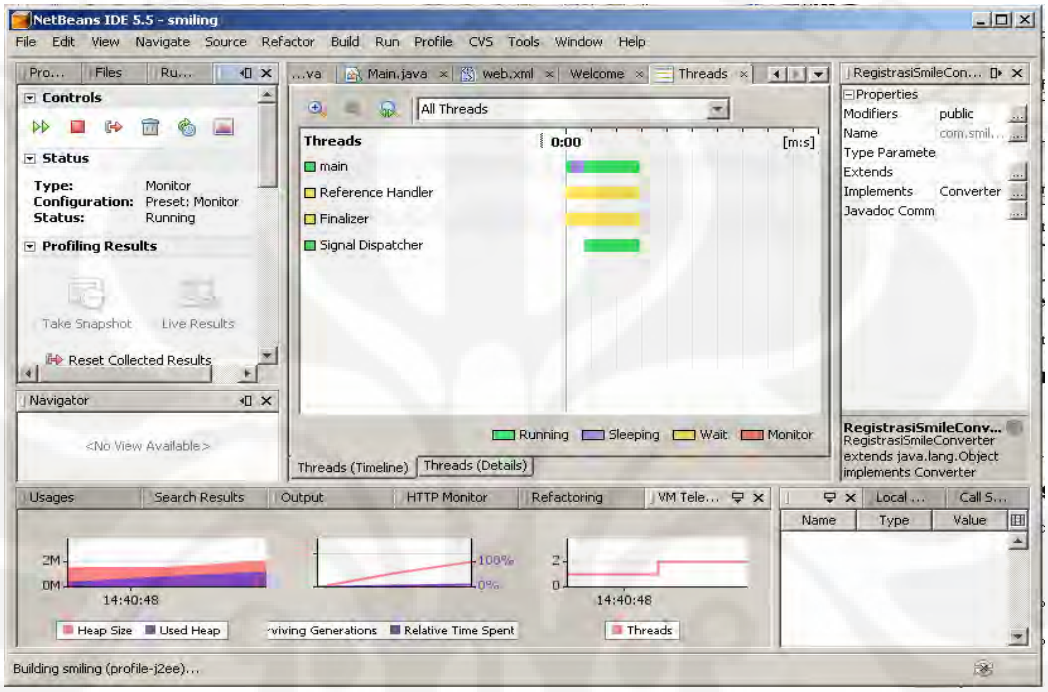


10. Proses “profile monitor” (4. *Monitoring Thread*)



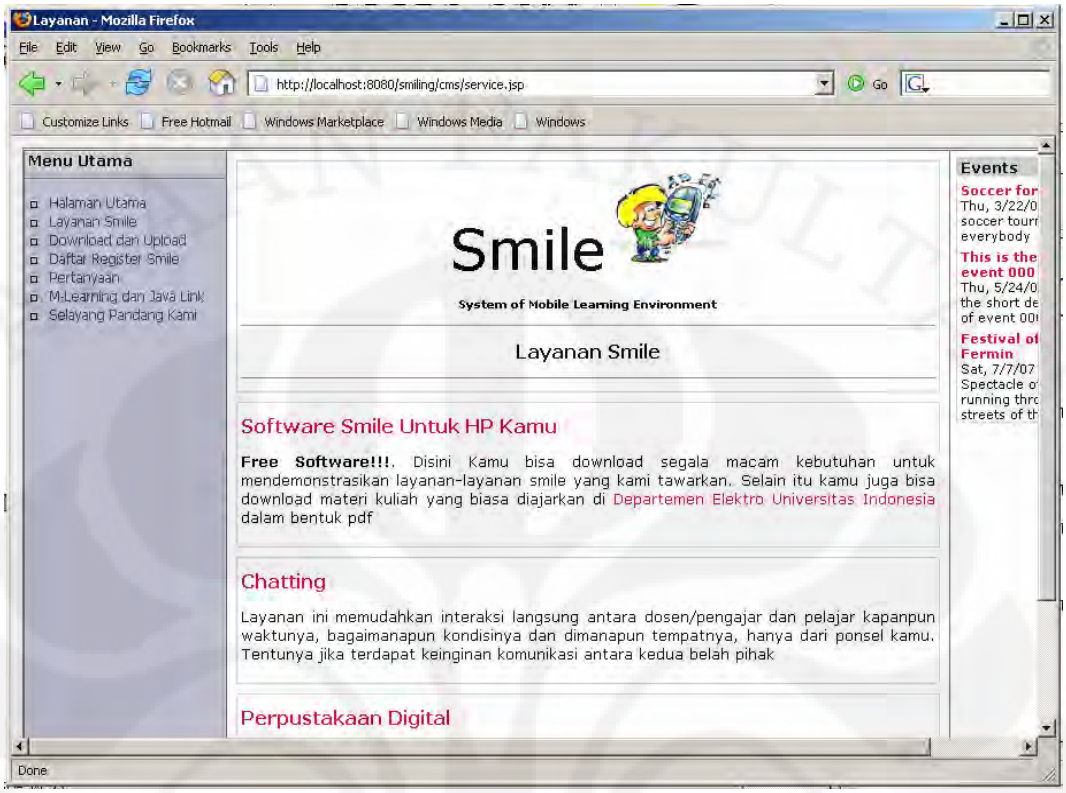
LAMPIRAN 2
LANGKAH MENGAMBIL DATA PERCOBAAN KETIKA MELAKUKAN *PROFILE*

1. *Build* SMiLE *Project*

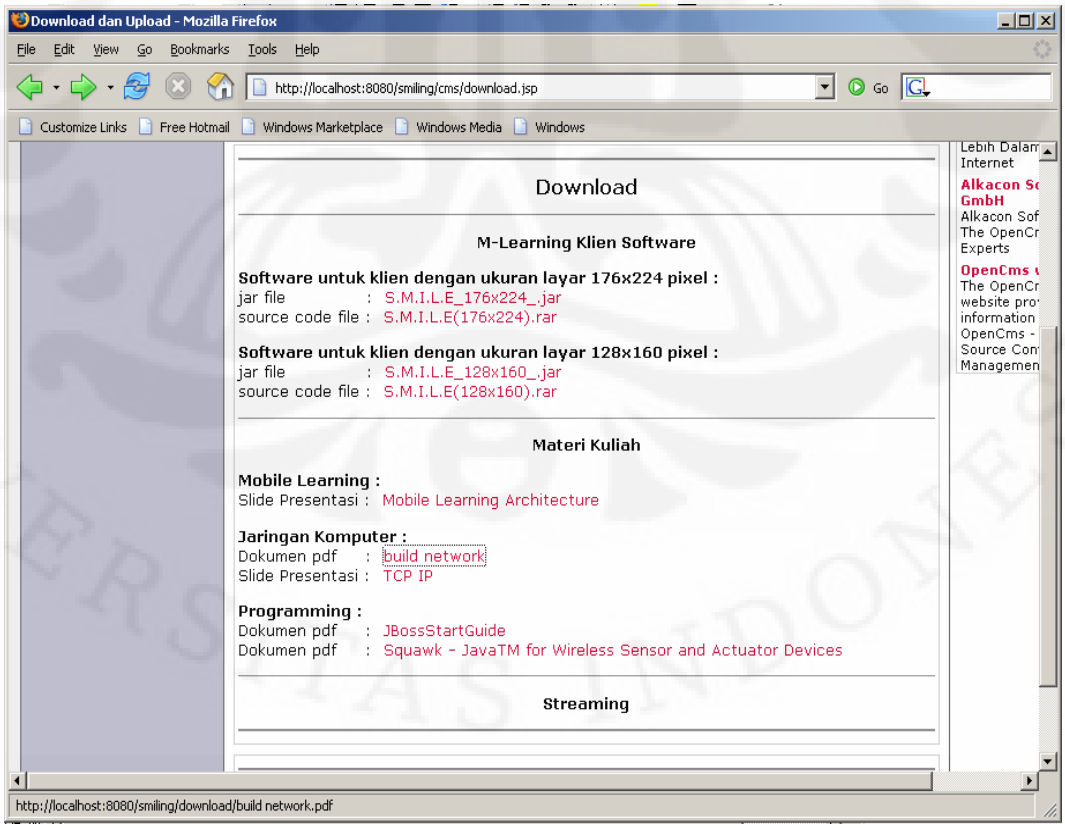


2. *Launching* SMiLE *in browser*

3. Akses "Layanan Smile" pada menu utama

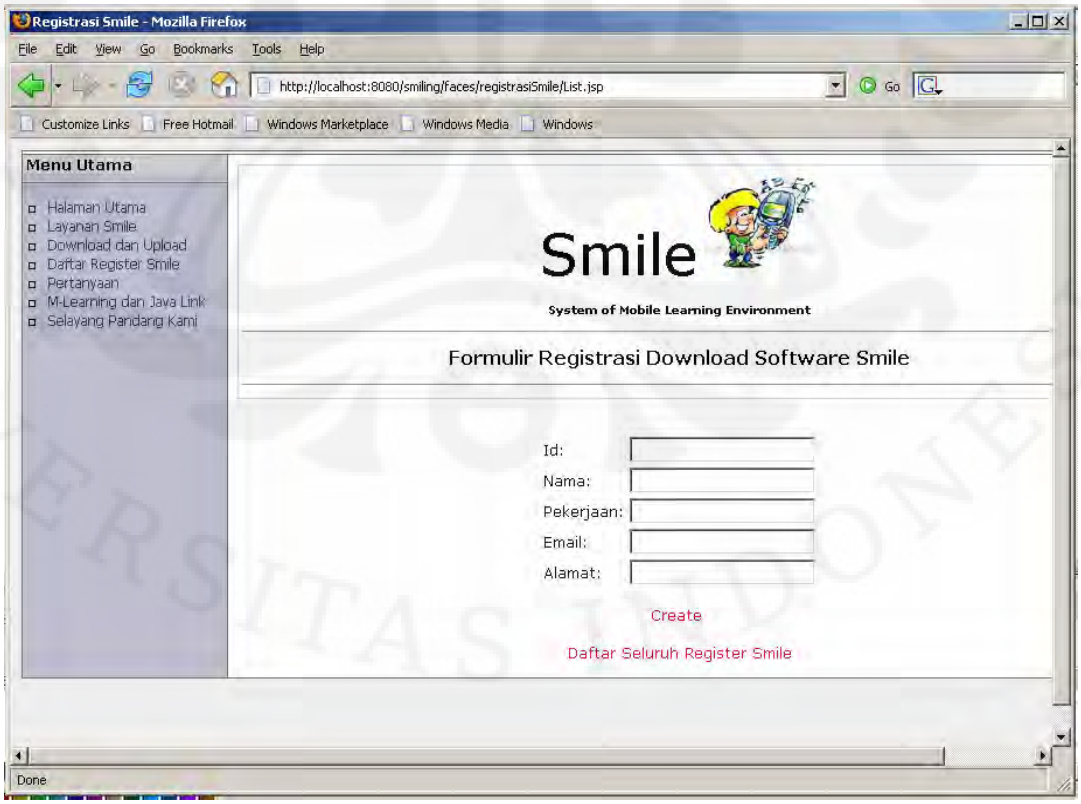


4. *Download file* pdf

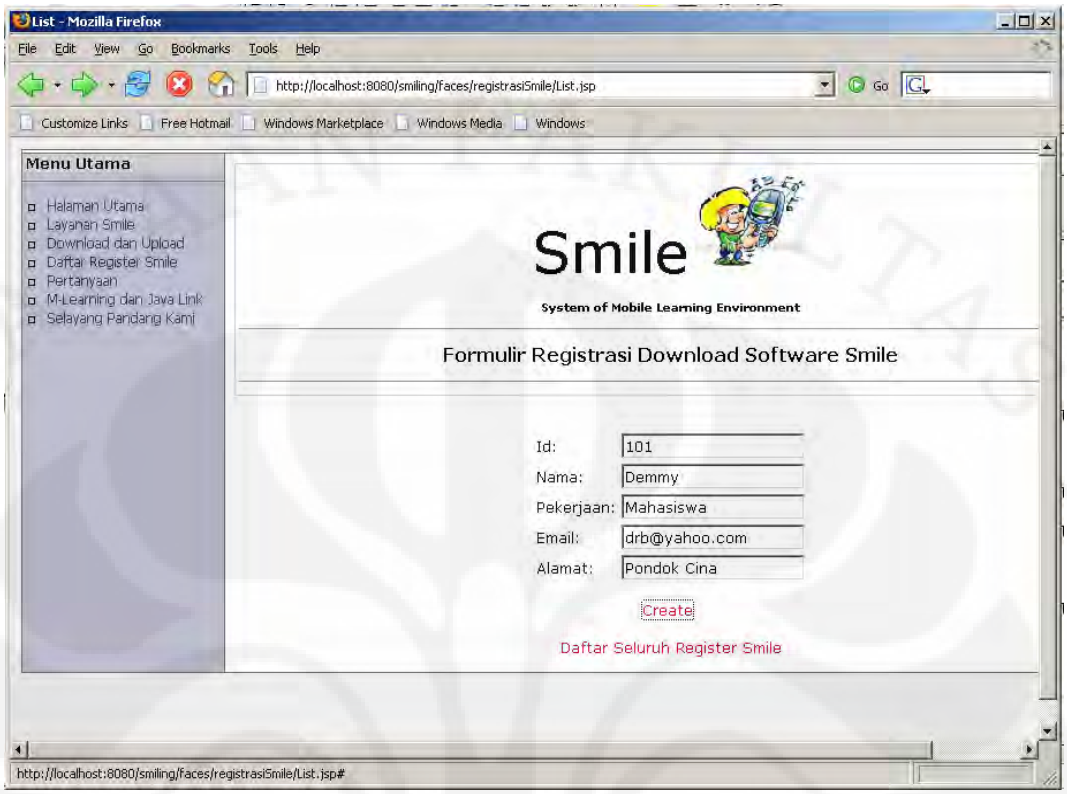


5. Akses "Daftar Registrasi Smile" pada menu utama

6. Klik *button* "Daftar"

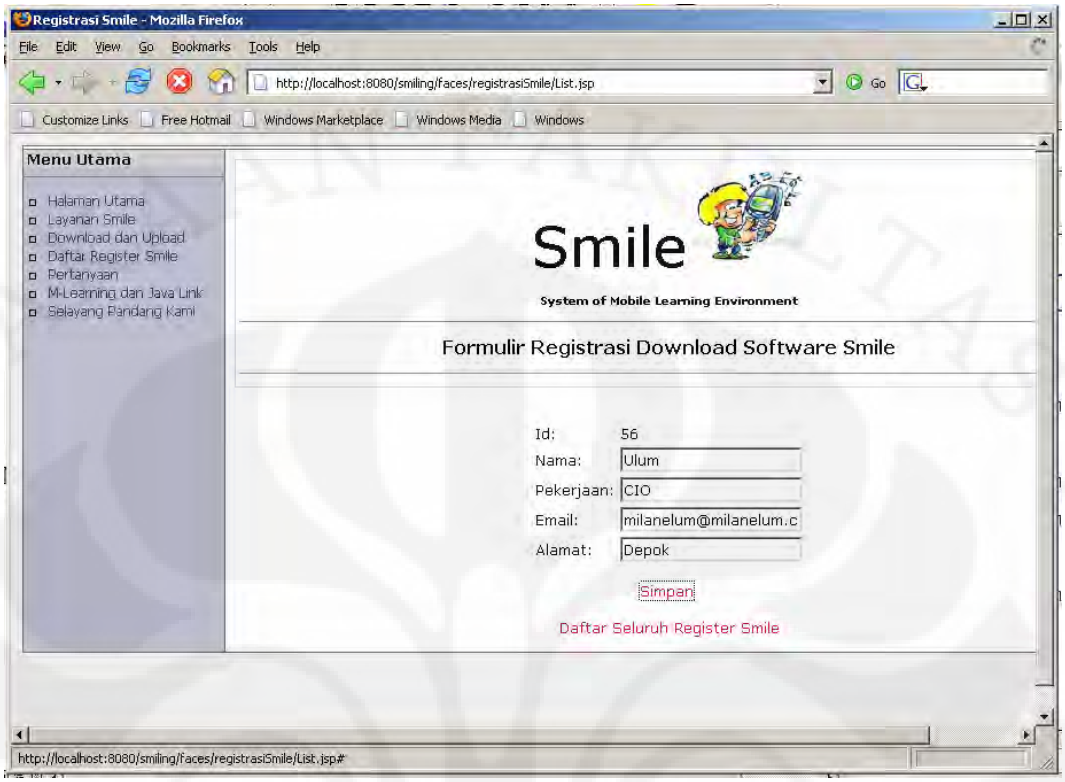


7. Klik *button* "Create"



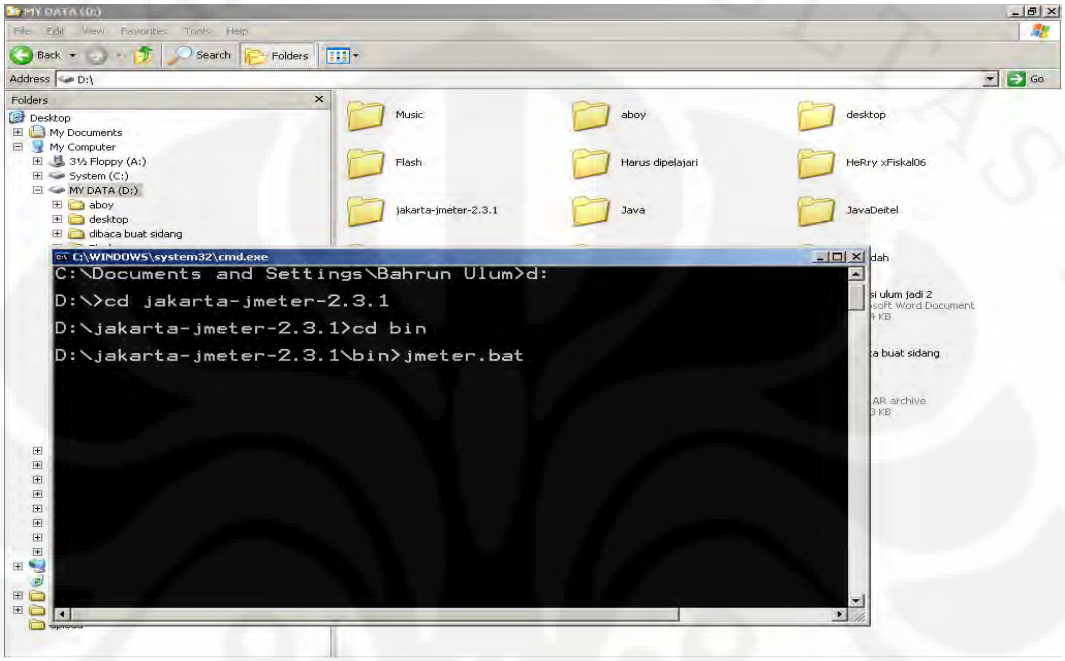
8. Klik *button* "Edit"

9. Klik *button* "Simpan"

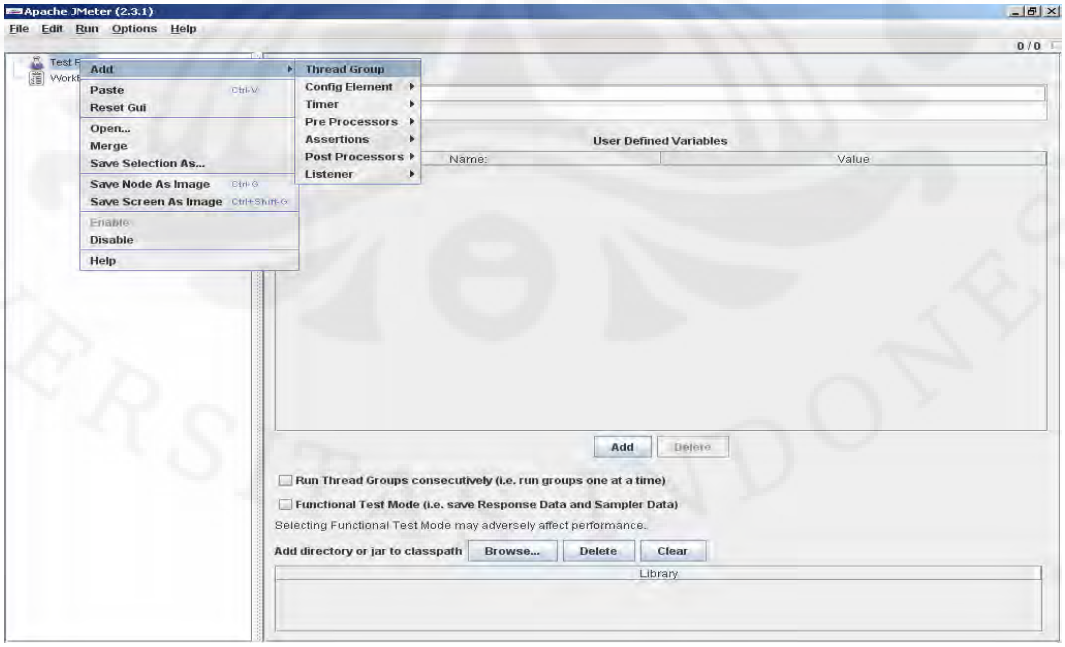


LAMPIRAN 3
LANGKAH MELAKUKAN UJI COBA *LOAD* DAN *SCALABILITY*

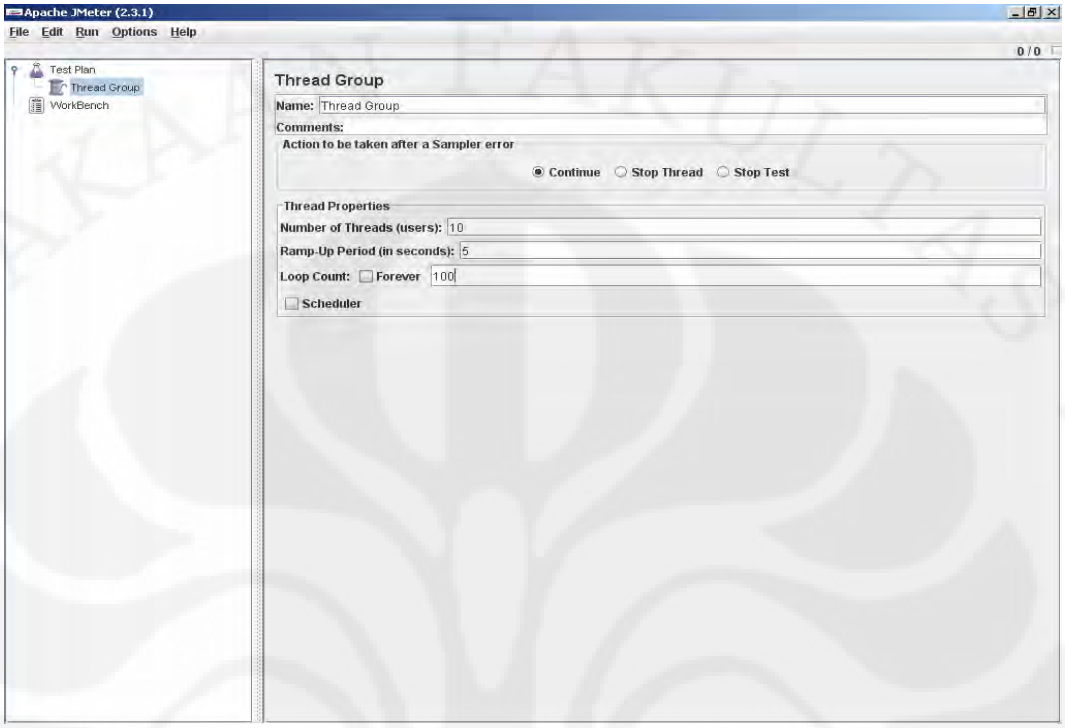
1. Jalankan JMeter.bat dari *folder* <Install Jakarta-JMeter-2.3.1>/bin



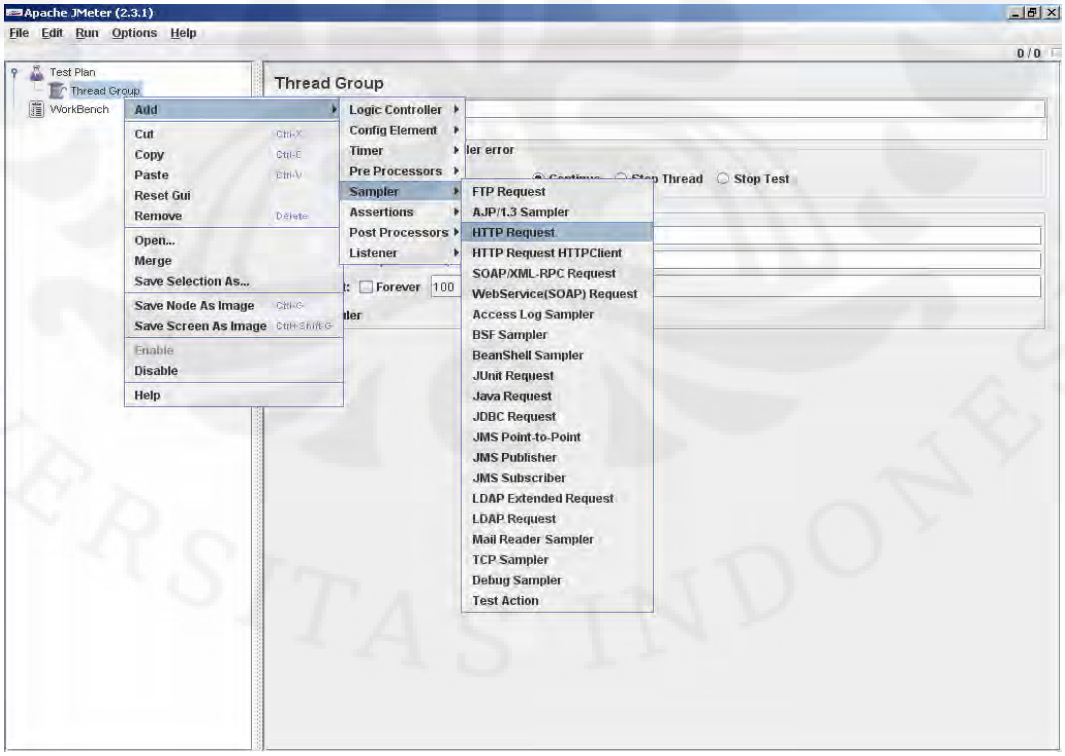
2. Setelah muncul *window* Apache JMeter, klik kanan "Test Plan", pilih "Add">"Thread Group".



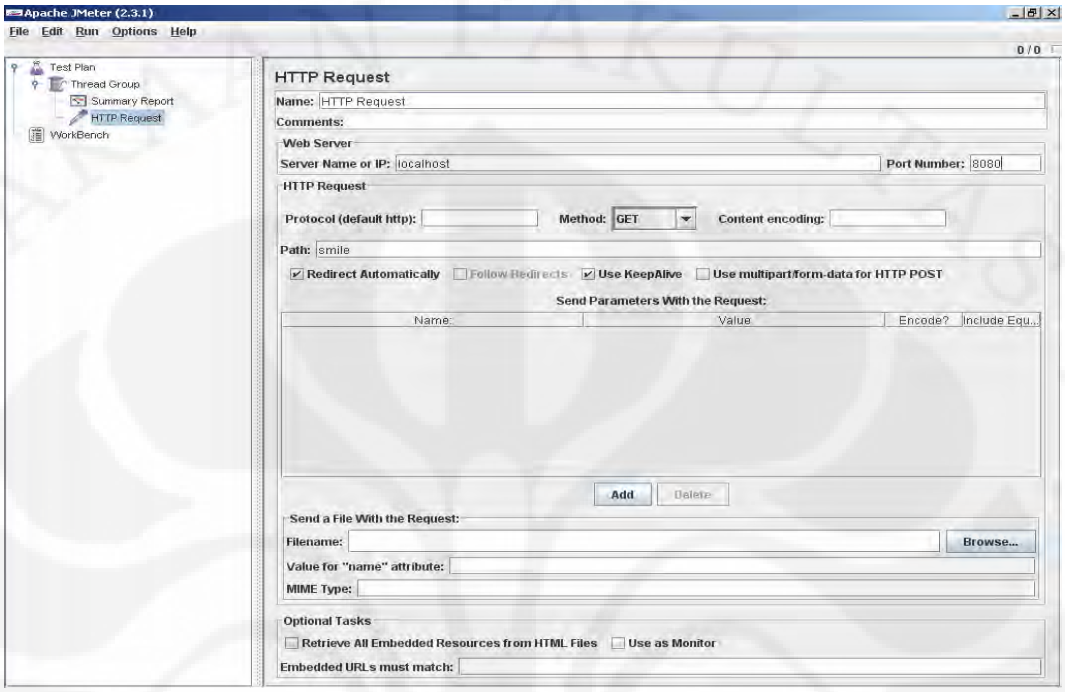
3. Kemudian masukkan nilai untuk “Number of Thread”, “Ramp-Up Periode” dan “Loop Count”.



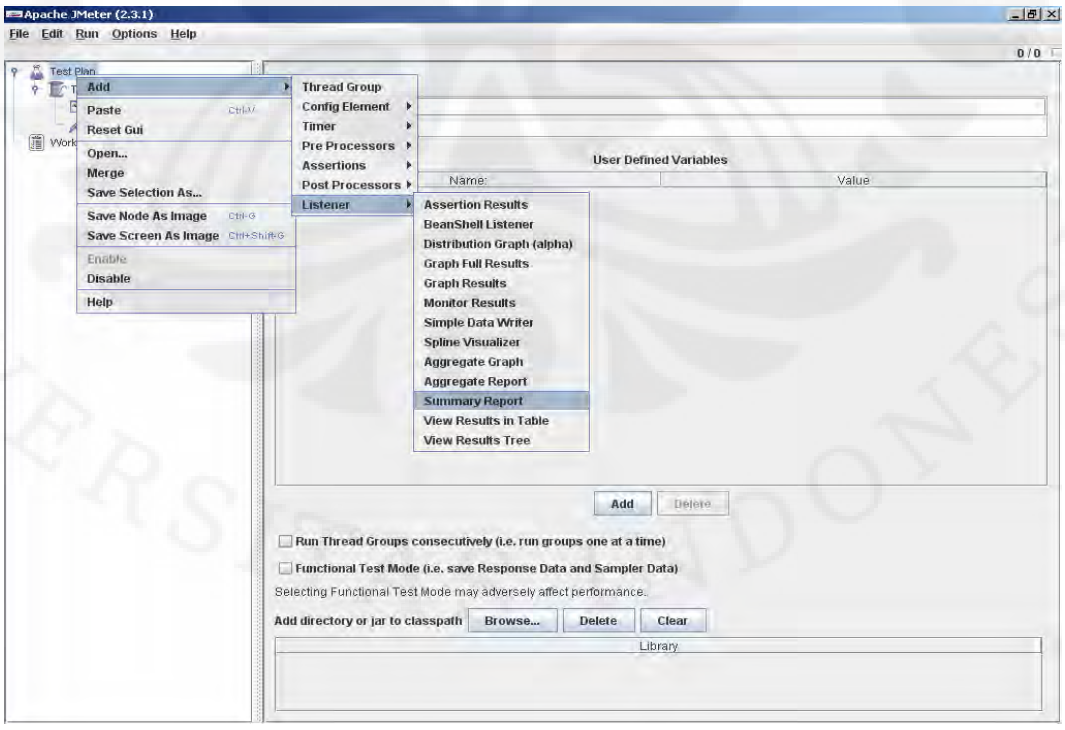
4. Klik kanan “Thread Group”, pilih “Add”> “Sampler” > “HTTP Request”.



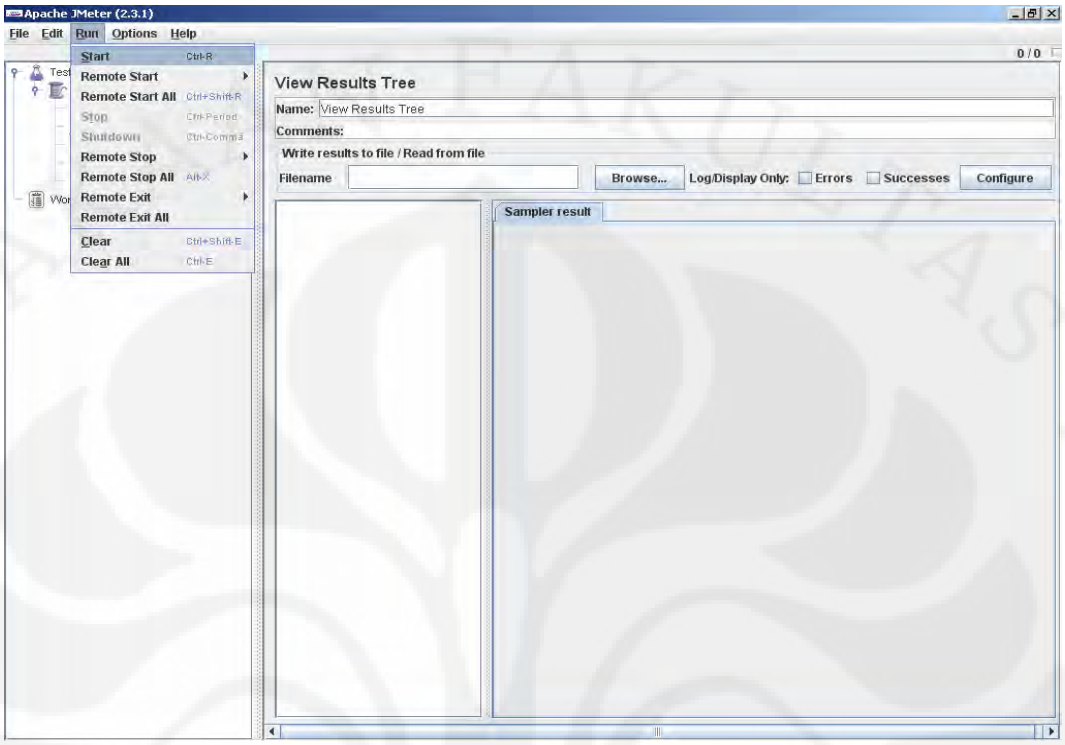
5. Isi “Server Name”, “Port Number” dan “Path”. Misal http://localhost:8080/smile.



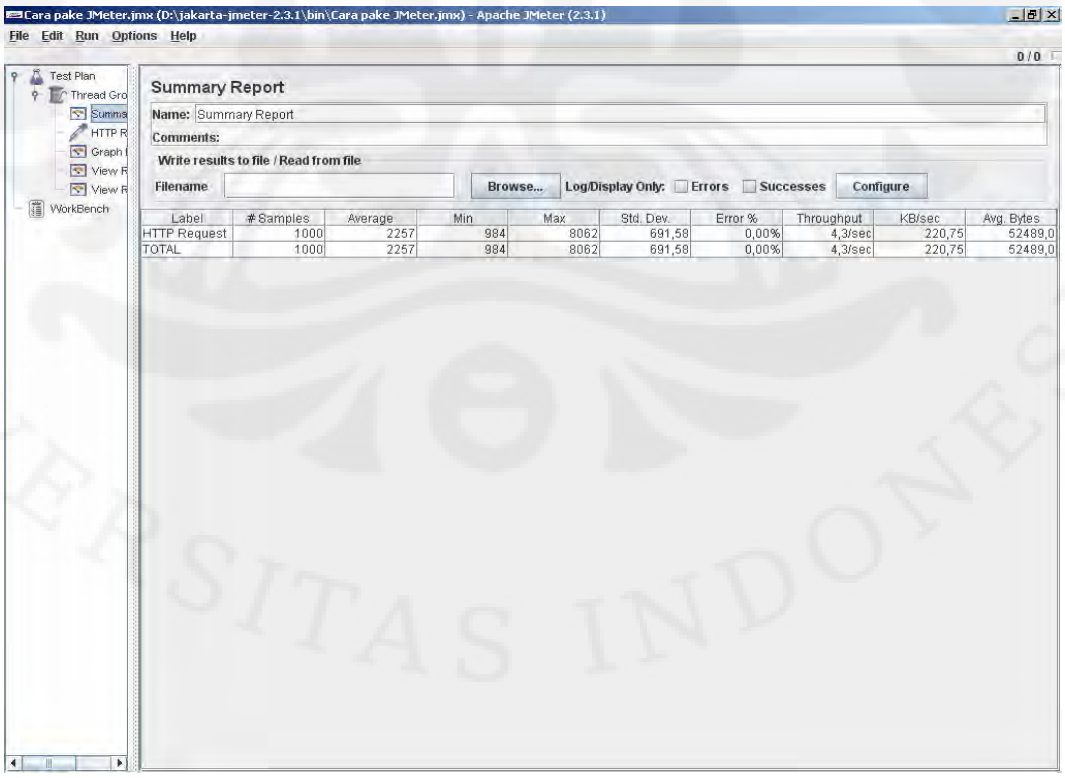
6. Klik kanan “Test Plan”, pilih “Add”> “Listener”, kemudian pilih jenis data yang ingin ditampilkan.



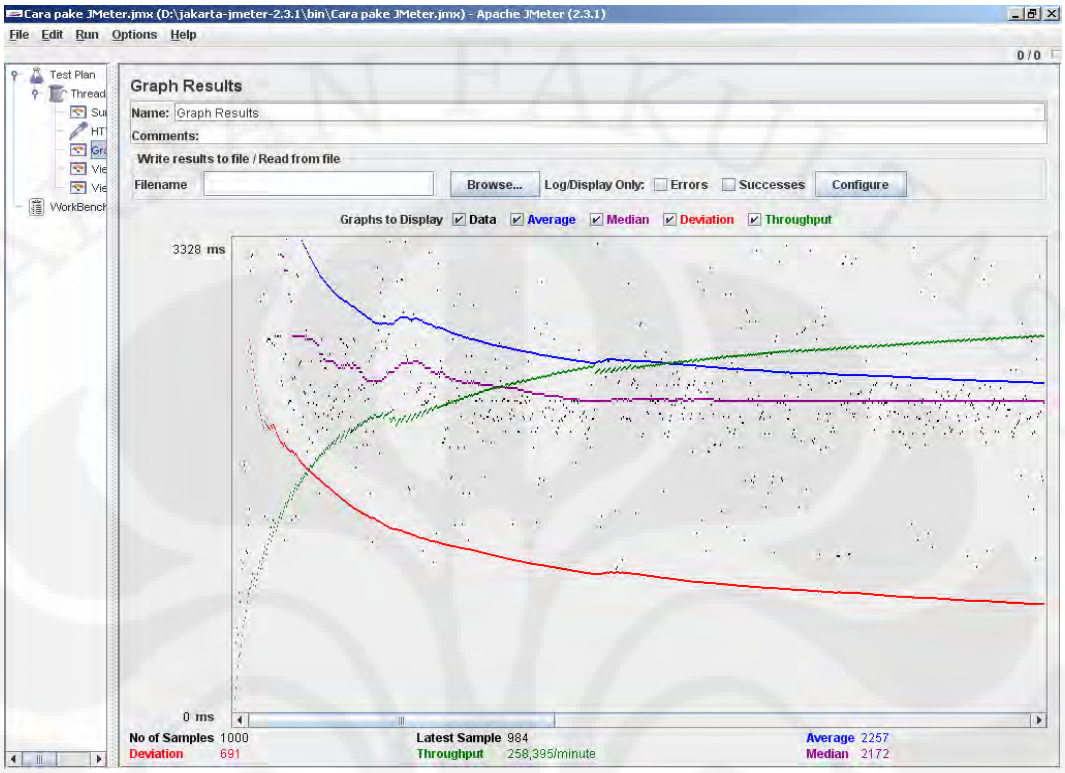
7. Klik menu utama ”Run”, kemudian pilih ”Start”.



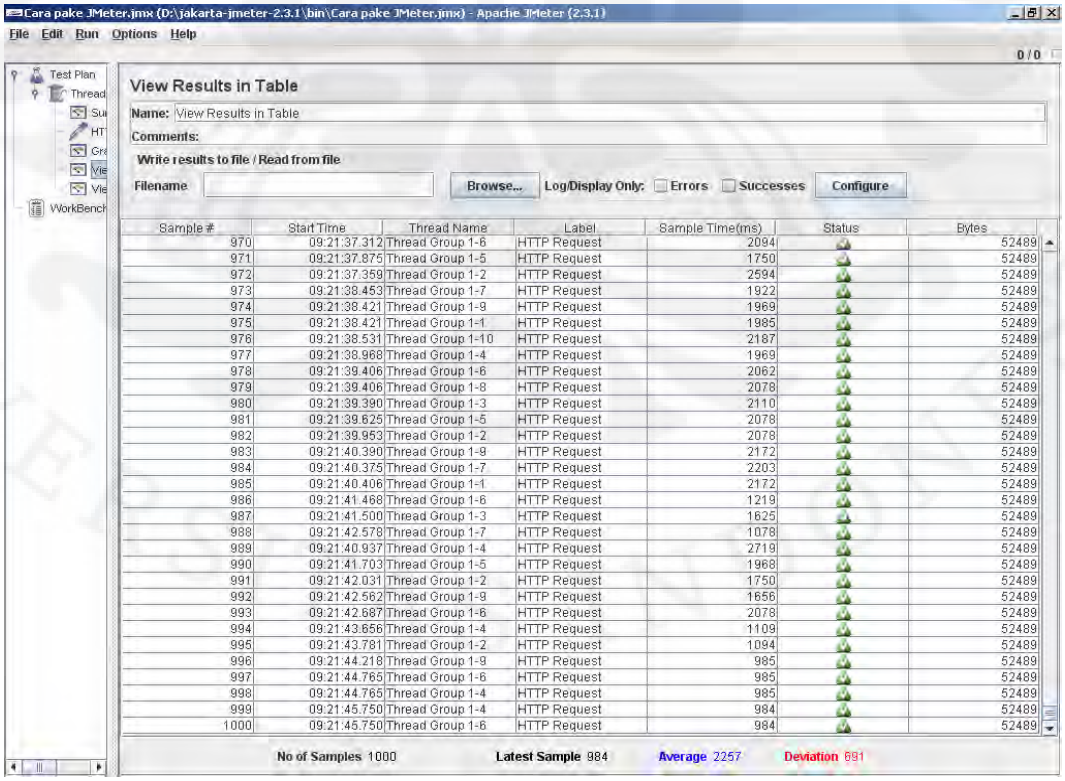
8. Hasil Uji Coba *Load* dan *Scalability*. (*Summary Report*).



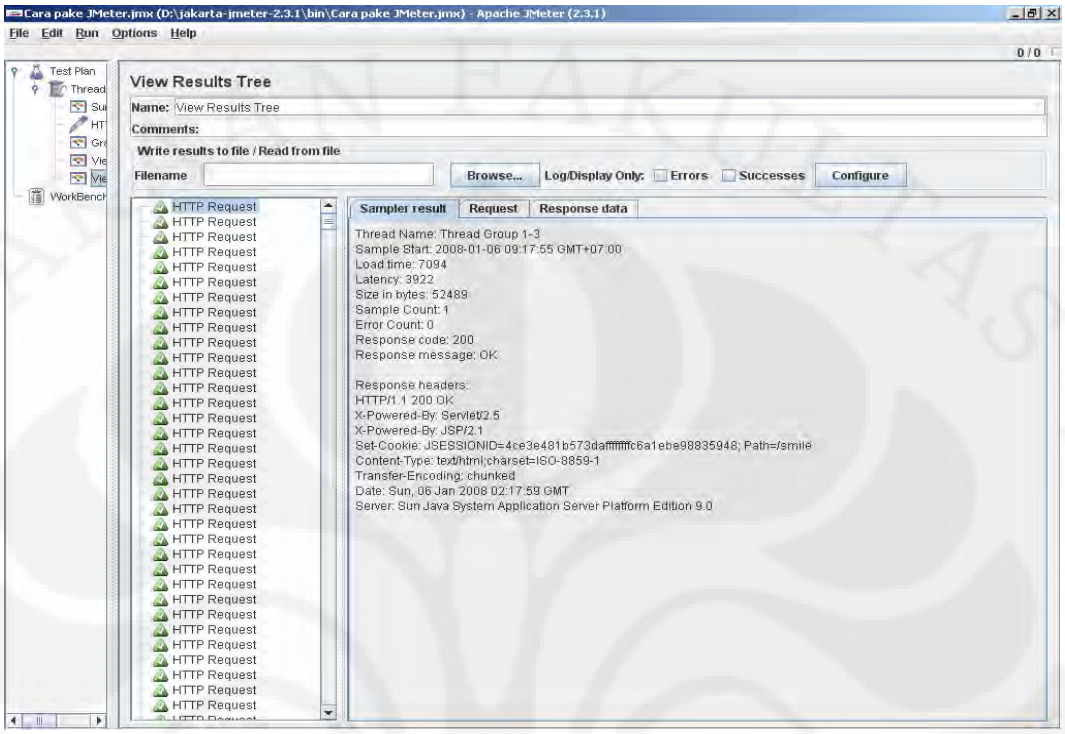
9. Hasil Uji Coba *Load* dan *Scalability*. (*Graph Result*).



10. Hasil Uji Coba *Load* dan *Scalability*. (*View Result in Table*).



11. Hasil Uji Coba *Load* dan *Scalability*. (*View Result Tree*).



LAMPIRAN 4
SOFTWARE YANG DIGUNAKAN DALAM MEMBUAT SKRIPSI INI

1. **Java EE 5.0 (www.sun.com)**
2. **Netbeans IDE 5.5 + Profiler 5.5 + Mobility Pack 5.5 (www.netbeans.org)**
3. **OpenCms 7.0.1 (www.Alkacoon)**
4. **JavaLibrary (Lim Song Jing)**
5. **WebJabber**

LAMPIRAN 5
KODE PROGRAM YANG DIJELASKAN PADA BAB 3

1. List.jsp

```
<f:view>
     <h:messages errorStyle="color: red" infoStyle="color: green" layout="table"/>
         <h:form>
              <p align="center"
             <h:dataTable value='#{registrasiSmile.registrasiSmiles}' var='item'
       border="1" cellpadding="2" cellspacing="0">
                        <h:column>
                            <f:facet name="header">
                               <h:outputText value="Id"/>
                             </f:facet>
                             <h:commandLink action="#{registrasiSmile.detailSetup}"
value="#{item.id}"/>
                          </h:column>
                          <%--
                          <h:column>
                             <f:facet name="header">
                                 <h:outputText value="Username"/>
                             </f:facet>
                             <h:outputText value="#{item.username}"/>
                           </h:column>
                           <h:column>
                              <f:facet name="header">
                                  <h:outputText value="Password"/>
                              </f:facet>
                              <h:outputText value="#{item.password}"/>
                           </h:column>
                            --%>
                           <h:column>
                               <f:facet name="header">
                                    <h:outputText value="Nama"/>
                               </f:facet>
                     <h:outputText value="#{item.nama}"/>
                           </h:column>
                           <h:column>
                               <f:facet name="header">
                                   <h:outputText value="Status"/>
                               </f:facet>
                               <h:outputText value="#{item.status}"/>
                            </h:column>
                            <h:column>
                                <f:facet name="header">
                                    <h:outputText value="Email"/>
                                </f:facet>
                                <h:outputText value="#{item.email}"/>
                             </h:column>
                             <h:column>
                                 <f:facet name="header">
                                      <h:outputText value="Alamat"/>
                                 </f:facet>
                                 <h:outputText value="#{item.alamat}"/>
                              </h:column>
                              <h:column>
                                   <h:commandLink value="Destroy"
action="#{registrasiSmile.destroy}">
                                        <f:param name="id" value="#{item.id}"/>
                                   </h:commandLink>
                                   <h:outputText value=" "/>
                                   <h:commandLink value="Edit"
action="#{registrasiSmile.editSetup}">
                                        <f:param name="id" value="#{item.id}"/>
                                   </h:commandLink>
                               </h:column>
               </h:dataTable>
```

```
                        <h:outputText value="Item #{registrasiSmile.firstItem +
1}..#{registrasiSmile.lastItem} of
                                    #{registrasiSmile.itemCount}"/> 
                <h:commandLink action="#{registrasiSmile.prev}" value="Previous
        #{registrasiSmile.batchSize}"
                                    rendered="#{registrasiSmile.firstItem >=
                        registrasiSmile.batchSize}"/> 
                         <h:commandLink action="#{registrasiSmile.next}"
value="Next #{registrasiSmile.batchSize}"
                                    rendered="#{registrasiSmile.lastItem +
                        registrasiSmile.batchSize
                        <=registrasiSmile.itemCount}"/> 
                <h:commandLink action="#{registrasiSmile.next}" value="Remaining
                #{registrasiSmile.itemCount –
                                    registrasiSmile.lastItem}"
                        rendered="#{registrasiSmile.lastItem <
                        registrasiSmile.itemCount &&
                                    registrasiSmile.lastItem + registrasiSmile.batchSize
                        > registrasiSmile.itemCount}"/>
                  </p>
                  <p align="center">
                           <h:commandLink action="#{registrasiSmile.createSetup}"
value="Registrasi disini"/>
                  </p>
         </h:form>
</f:view>
```

2. RegistrasiSmileController.java

```
package com.smile.logic;

import com.smile.entity.RegistrasiSmile;
import java.util.ArrayList;
import java.util.Collection;
import javax.annotation.Resource;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.PersistenceUnit;
import javax.persistence.Query;
import javax.transaction.UserTransaction;

/**
 *
 * @author Bahrun Ulum
 */
public class RegistrasiSmileController {

    /** Creates a new instance of RegistrasiSmileController */
    public RegistrasiSmileController() {
    }

    private RegistrasiSmile registrasiSmile;

    private DataModel model;

    @Resource
    private UserTransaction utx;

    @PersistenceUnit(unitName = "smilePU")
    private EntityManagerFactory emf;

    private EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    private int batchSize = 20;

    private int firstItem = 0;
```

```
public RegistrasiSmile getRegistrasiSmile() {
    return registrasiSmile;
}

public void setRegistrasiSmile(RegistrasiSmile registrasiSmile) {
    this.registrasiSmile = registrasiSmile;
}

public DataModel getDetailRegistrasiSmiles() {
    return model;
}

public void setDetailRegistrasiSmiles(Collection<RegistrasiSmile> m) {
    model = new ListDataModel(new ArrayList(m));
}

public String createSetup() {
    this.registrasiSmile = new RegistrasiSmile();
    return "registrasiSmile_create";
}

public String create() {
    EntityManager em = getEntityManager();
    try {
        utx.begin();
        em.persist(registrasiSmile);
        utx.commit();
        addSuccessMessage("RegistrasiSmile was successfully created.");
    } catch (Exception ex) {
        try {
            addErrorMessage(ex.getLocalizedMessage());
            utx.rollback();
        } catch (Exception e) {
            addErrorMessage(e.getLocalizedMessage());
        }
    } finally {
        em.close();
    }
    return "registrasiSmile_list";
}

public String detailSetup() {
    setRegistrasiSmileFromRequestParam();
    return "registrasiSmile_detail";
}

public String editSetup() {
    setRegistrasiSmileFromRequestParam();
    return "registrasiSmile_edit";
}

public String edit() {
    EntityManager em = getEntityManager();
    try {
        utx.begin();
        registrasiSmile = em.merge(registrasiSmile);
        utx.commit();
        addSuccessMessage("RegistrasiSmile was successfully updated.");
    } catch (Exception ex) {
        try {
            addErrorMessage(ex.getLocalizedMessage());
            utx.rollback();
        } catch (Exception e) {
            addErrorMessage(e.getLocalizedMessage());
        }
    } finally {
        em.close();
    }
    return "registrasiSmile_list";
}

public String destroy() {
    EntityManager em = getEntityManager();
```

```
        try {
            utx.begin();
            RegistrasiSmile registrasiSmile =
getRegistrasiSmileFromRequestParam();
            registrasiSmile = em.merge(registrasiSmile);
            em.remove(registrasiSmile);
            utx.commit();
            addSuccessMessage("RegistrasiSmile was successfully deleted.");
        } catch (Exception ex) {
            try {
                addErrorMessage(ex.getLocalizedMessage());
                utx.rollback();
            } catch (Exception e) {
                addErrorMessage(e.getLocalizedMessage());
            }
        } finally {
            em.close();
        }
        return "registrasiSmile_list";
    }

    public RegistrasiSmile getRegistrasiSmileFromRequestParam() {
        EntityManager em = getEntityManager();
        try{
            RegistrasiSmile o = (RegistrasiSmile) model.getRowData();
            o = em.merge(o);
            return o;
        } finally {
            em.close();
        }
    }

    public void setRegistrasiSmileFromRequestParam() {
        RegistrasiSmile registrasiSmile = getRegistrasiSmileFromRequestParam();
        setRegistrasiSmile(registrasiSmile);
    }

    public DataModel getRegistrasiSmiles() {
        EntityManager em = getEntityManager();
        try{
            Query q = em.createQuery("select object(o) from RegistrasiSmile as
o");
            q.setMaxResults(batchSize);
            q.setFirstResult(firstItem);
            model = new ListDataModel(q.getResultList());
            return model;
        } finally {
            em.close();
        }
    }

    public static void addErrorMessage(String msg) {
        FacesMessage facesMsg = new FacesMessage(FacesMessage.SEVERITY_ERROR, msg,
msg);
        FacesContext fc = FacesContext.getCurrentInstance();
        fc.addMessage(null, facesMsg);
    }

    public static void addSuccessMessage(String msg) {
        FacesMessage facesMsg = new FacesMessage(FacesMessage.SEVERITY_INFO, msg,
msg);
        FacesContext fc = FacesContext.getCurrentInstance();
        fc.addMessage("successInfo", facesMsg);
    }

    public RegistrasiSmile findRegistrasiSmile(Long id) {
        EntityManager em = getEntityManager();
        try{
            RegistrasiSmile o = (RegistrasiSmile) em.find(RegistrasiSmile.class,
id);
            return o;
        } finally {
            em.close();
        }
```

```
    }

    public int getItemCount() {
        EntityManager em = getEntityManager();
        try{
            int count = ((Long) em.createQuery("select count(o) from
RegistrasiSmile as o").getSingleResult()).intValue();
            return count;
        } finally {
            em.close();
        }
    }

    public int getFirstItem() {
        return firstItem;
    }

    public int getLastItem() {
        int size = getItemCount();
        return firstItem + batchSize > size ? size : firstItem + batchSize;
    }

    public int getBatchSize() {
        return batchSize;
    }

    public String next() {
        if (firstItem + batchSize < getItemCount()) {
            firstItem += batchSize;
        }
        return "registrasiSmile_list";
    }

    public String prev() {
        firstItem -= batchSize;
        if (firstItem < 0) {
            firstItem = 0;
        }
        return "registrasiSmile_list";
    }

}
```

3. RegistrasiSmile.java

```
package com.smile.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 * Entity class RegistrasiSmile
 *
 * @author Bahrun Ulum
 */
@Entity
@Table(name="REGISTRASI_SMILE")
public class RegistrasiSmile implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(name = "USERNAME", nullable = false)
    private String username;

    @Column(name = "PASSWORD", nullable = false)
```

```
    private String password;

    @Column(name = "NAMA")
    private String nama;

    @Column(name = "STATUS", nullable = false)
    private String status;

    @Column(name = "EMAIL")
    private String email;

    @Column(name = "ALAMAT")
    private String alamat;

    /** Creates a new instance of RegistrasiSmile */
    public RegistrasiSmile() {
    }

    /**
     * Gets the id of this RegistrasiSmile.
     * @return the id
     */
    public Long getId() {
        return this.id;
    }

    /**
     * Sets the id of this RegistrasiSmile to the specified value.
     * @param id the new id
     */
    public void setId(Long id) {
        this.id = id;
    }

    /**
     * Returns a hash code value for the object.  This implementation computes
     * a hash code value based on the id fields in this object.
     * @return a hash code value for this object.
     */
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }

    /**
     * Determines whether another object is equal to this RegistrasiSmile.  The
result is
     * <code>true</code> if and only if the argument is not null and is a
RegistrasiSmile object that
     * has the same id field values as this object.
     * @param object the reference object with which to compare
     * @return <code>true</code> if this object is the same as the argument;
     * <code>false</code> otherwise.
     */
    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are
not set
        if (!(object instanceof RegistrasiSmile)) {
            return false;
        }
        RegistrasiSmile other = (RegistrasiSmile)object;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id)))
return false;
        return true;
    }

    /**
     * Returns a string representation of the object.  This implementation
constructs
     * that representation based on the id fields.
     * @return a string representation of the object.
```

```
     */
    @Override
    public String toString() {
        return "com.smile.entity.RegistrasiSmile[id=" + id + "]";
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getNama() {
        return nama;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getAlamat() {
        return alamat;
    }

    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }

}
```