IMPLEMENTASI ARSITEKTUR MODEL 2X PADA STUDI KASUS SITUS PENGARSIPAN TUGAS AKHIR MENGGUNAKAN TEKNOLOGI JSF DAN XML/XSLT DENGAN ORACLE TEXT UNTUK PENYIMPANAN DOKUMEN

TUGAS AKHIR

Disusun oleh :

HAIKAL
5100 100 045

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2004

IMPLEMENTASI ARSITEKTUR MODEL 2X PADA STUDI KASUS SITUS PENGARSIPAN TUGAS AKHIR MENGGUNAKAN TEKNOLOGI JSF DAN XML/XSLT DENGAN ORACLE TEXT UNTUK PENYIMPANAN DOKUMEN

TUGAS AKHIR

Diajukan Guna Memperoleh Sebagian Persyaratan
Untuk memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Rully Soelaiman, S.Kom	Darlis Heru Murti, S.Kom
NIP. 132.086.802	NIP. 132.306.430

SURABAYA
2004

ABSTRAK

Perkembangan teknologi aplikasi web semakin terstruktur dengan adanya paradigma pengembangan aplikasi web berarsitektur *Model-View-Controller* (MVC). Melalui implementasi arsitektur MVC, proses pengembangan aplikasi mengalami pemisahan yang jelas antara pengembangan fungsi proses bisnis dan pengembangan fungsi antarmuka. Teknologi JavaServer Faces (JSF) memiliki peran yang tidak sedikit dalam memberikan standar bersama untuk implementasi arsitektur MVC ini. Tidak hanya itu, JSF juga memberikan standar baru berupa pengembangan antarmuka aplikasi yang modular. Mengikuti berkembangnya teknologi format data *eXtensible Markup Language* (XML), dikenal arsitektur Model 2X yang mengadopsi kelebihan-kelebihan XML pada arsitektur MVC. Dengan Model 2X arsitektur MVC memiliki kemampuan untuk mengolah data keluaran melalui transformasi XML pada layer presentasi secara modular.

Pada Tugas Akhir ini dilakukan implementasi arsitektur Model 2X dengan studi kasus pengarsipan tugas akhir. Implementasi arsitektur ini diharapkan dapat memberikan contoh konkrit bagaimana arsitektur ini memberikan fleksibilitas layer presentasi berikut pemisahan antara fungsi proses bisnis dan fungsi antarmuka aplikasi. Untuk mengatur penyimpanan dokumen terformat, aplikasi pengarsipan tugas akhir menggunakan teknologi Oracle Text. Selain untuk menyimpan dokumen pada database, teknologi ini juga memberikan kemampuan pencarian pada isi dokumen terformat.

Dari hasil uji coba, aplikasi ini dapat mengimplementasikan arsitektur Model 2X dengan baik. Aplikasi ini memiliki layer presentasi yang fleksibel, disertai pemisahan fungsi proses bisnis dan antarmuka. Untuk fungsi pencarian, aplikasi ini dapat melakukan pencarian kata pada dokumen terformat dengan benar.

KATA PENGANTAR

Alhamdulillaah, segala puja dan puji hanyalah bagi Allah Yang SWT, Yang telah melimpahkan hidayah dan kasih sayang-Nya. Atas kemudahan yang diberikan-Nya lah penulis dapat menyelesaikan tugas akhir ini. *Shalawat* serta salam tetap kita curahkan kepada *Rasulullah* Muhammad SAW.

Penulis memberi judul pada tugas akhir ini: **IMPLEMENTASI ARSITEKTUR MODEL 2X PADA STUDI KASUS SITUS PENGARSIPAN TUGAS AKHIR MENGGUNAKAN TEKNOLOGI JSF DAN XML/XSLT DENGAN ORACLE TEXT UNTUK PENYIMPANAN DOKUMEN.**

Penulis berharap semoga tugas akhir ini dapat membantu pengembangan teknologi informasi di jurusan Teknik Informatika ITS. Tugas akhir ini masih jauh dari sempurna, oleh karena itu sangat diharapkan segala masukan bagi pengembangan selanjutnya

Akhirnya, penulis mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu penyelesaian tugas akhir ini.

Surabaya, Agustus 2004

Haikal

UCAPAN TERIMA KASIH

Alhamdulillah, penulis tidak akan dapat menyelesaikan tugas akhir ini tanpa pertolongan Allah SWT. Melalui kesempatan ini penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Allah SWT, Rabbul izzati yang telah memberikan limpahan rahmat dan hidayah-Nya, serta segala nikmat yang tak terhitung jumlahnya.
2. Ayahanda dan Ibunda penulis, semoga Allah melimpahkan ampunan, kasih sayang dan hidayah-Nya kepada beliau berdua.
3. Prof. Dr. Ir. Arif Djunaidy, M.Sc, selaku Ketua Dekan Fakultas Teknologi Informasi.
4. Bapak Yudhi Purwananto, M.Kom, selaku Ketua Jurusan Teknik Informatika.
5. Bapak Ir. Khakim GhozaliI, selaku Dosen wali penulis.
6. Bapak Rully Soelaiman, S Kom dan Bapak Darlis, S.Kom yang telah memberikan berjuta-juta bimbingan yang tak ternilai harganya.
7. Segenap dosen pengajar dan karyawan Fakultas Teknologi Informasi.
8. Kak Anis, Kak Acha, dan Bang Zaki, semoga Allah memudahkan kita semua dalam segala urusan. *Kapan ya... ngumpul lagi??*
9. Angkatan C10, Ichsan (*San, gpp ya, aku duluan...),* Taufan, Dimas, Joko, Jakka, Gunna(*yg bentar lagi jadi ayah...*), Angga, Deka, Tri, Khoirul, David, Toni, Adit, Lutfi, Eggy, Tyarso, pokoknya semuanya

deh, terimakasih atas pengalaman empat tahun (dan masih terus berlanjut....) yg tak ternilai harganya.

10. Rekan-rekan di Lab SI, Macan Putih, Mas Agung(*Cepetan lho mas...*), Fauzan, Angga, Mumud, *and the others*, makasih banyak ya.....
11. Rekan-rekan Lab Prog, Kris, Wawan, Iwan, Kholimi dan yang lainnya makasih ya... udah mbantuin repot-repot urusan TA.
12. Teman-teman kost, Mas Triandi, Mas Habib, Mas Nanang, Mas Irul, Mas Eko, Erik, Anang, Hasan, Mas Hanif, Anas, Mas Gatot, Yuyut, Yanu, Triono, Mas Agung, dan yang lainnya juga. Terima kasih semua atas kebersamaannya.
13. Adik-adikku Salman, Rahmat, Izzam, Aris, semuanya. Cepetan nyusul ya... ☺.
14. Mas Bhakti, Mas Ary MS, Mas Anang, Mas Kendy, Mas Guntar (*printernya barokah lho mas...*), Toni, Mas Liga, JePe (*migrasi j-3!!!*), Kamal (*dah anti klimaks kan...???*), Younan, Udin, Hendra, makasih *buanyak* atas bantuannya ..... semoga Allah SWT membalas jasa-jasa kalian.
15. Rekan-rekan IRC, Dimas (*makasih atas bantuan do'anya*), Mas Irfan, Mas Osa, dan yang lainnya... keep fight!!!
16. Kepada saudara-saudaraku dan teman-teman sekampus yang tidak mungkin disebutkan satu persatu, terima kasih semua.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

BAB I

PENDAHULUAN

Bab ini berisi penjelasan secara umum tentang tugas akhir yang dikerjakan. Meliputi latar belakang tugas akhir, permasalahan yang dihadapi, batasan masalah pengerjaan tugas akhir, tujuan, metodologi pembuatan tugas akhir, serta sistematika pembahasan tugas akhir.

1.1. LATAR BELAKANG

Sebuah aplikasi merupakan perangkat lunak yang dikembangkan untuk memenuhi kebutuhan tertentu, misalnya sebuah *Word Processor* dikembangkan atas kebutuhan akan perangkat lunak untuk mempermudah pekerjaan-pekerjaan yang berkaitan dengan pengetikan. Aplikasi berbasis web adalah aplikasi yang diinstal pada sisi server web. Aplikasi ini diakses melalui internet dengan menggunakan browser. Untuk membangun sebuah aplikasi berbasis web, terdapat beberapa pilihan bahasa pemrograman yang dapat digunakan. Antara lain PHP, ASP, JSP, PL/SQL dan beberapa bahasa pemrograman lainnya.

JavaServer Pages (JSP) sebagai sebuah bahasa *scripting* dapat digunakan untuk membangun sebuah website yang dinamis dengan pemrograman di sisi server. Sebagai pengembangan dari servlet, maka spesifikasi JSP dibuat atas fungsionalitas servlet. Implementasi JSP pada pembuatan website dinamis memudahkan pengembang perangkat lunak dalam membangun antarmuka halaman web sehingga proses pembuatan website lebih mirip seperti mendesain

halaman web daripada membuat sebuah program java. JSP ideal untuk membuat sebuah halaman web dengan isi yang dinamis. Selain itu, JSP dapat menggunakan *JavaBeans* untuk memisahkan antara pembuatan isi halaman web dengan pemrograman Java. Di sini JSP berperan sebagai *presentation layer* aplikasi web.

Dalam hal pembuatan antarmuka, dikenal sebuah teknologi dari Java yang dapat diimplementasikan pada aplikasi JSP, yaitu JavaServer Faces (JSF). JSF adalah sebuah *framework* untuk membuat antarmuka pada aplikasi server berbasis web. Pada umumnya untuk menangani antarmuka, pengembang akan menangani secara langsung melalui kode-kode JSP. Untuk antarmuka yang sederhana, hal ini tidak akan mengurangi produktivitas pengembang. Namun jika antarmuka yang dibutuhkan semakin kompleks, maka pengembang harus memberikan konsentrasi lebih untuk membuat antarmuka tersebut.

Dengan model pengembangan yang disediakan teknologi JSF, pembuatan aplikasi web dapat dilakukan dengan lebih mudah melalui komponen-komponen antarmuka yang *reusable*, menghubungkan komponen-komponen tersebut dengan sumber data tertentu serta membuat penanganan even dari sisi server untuk menangani even-even komponen antarmuka dari *client*. Dengan teknologi ini pengembang perangkat lunak dapat memfokuskan pekerjaannya pada proses bisnis aplikasi, sedang kompleksitas antarmuka ditangani oleh JSF.

Untuk *presentation layer*, digunakan teknologi yang dapat memisahkan antara data yang akan ditampilkan dengan sisi desain tampilan halaman web, yaitu XML. XML dikembangkan untuk memenuhi kebutuhan akan sebuah format pertukaran data yang dapat diterima oleh semua vendor pengembang perangkat

lunak. Untuk tujuan inilah teknologi XML dikembangkan. Sehingga nantinya berbagai perangkat lunak yang dihasilkan oleh berbagai vendor dapat berkomunikasi, bertukar data melalui sebuah format data, dan dapat memproses data tersebut. Dalam format XML, data dapat disimpan dalam format yang netral dari sisi platform, dan juga netral dari sisi bahasa pemrograman. Dengan *Document Type Definition* yang merupakan salah satu bagian dari spesifikasi XML, format data XML dapat dikembangkan sesuai dengan kebutuhan tanpa kehilangan konsistensi format data.

Kombinasi teknologi JSF dalam pengembangan perangkat lunak dengan Model 2X diharapkan dapat menghasilkan sebuah perangkat lunak yang memiliki layer presentasi yang lebih modular dan fleksibel dibandingkan dengan Model 1 atau JSP pada umumnya. Sebagai studi kasus, dibuat sebuah aplikasi Pengarsipan Tugas Akhir. Adapun pertimbangan penulis memilih studi kasus ini karena adanya kebutuhan akan perangkat lunak yang mengatur pengarsipan tugas akhir hasil karya mahasiswa. Sehingga tugas akhir yang pernah dibuat oleh mahasiswa tersimpan dengan baik dan dapat ditelusuri kembali dengan cepat ketika dibutuhkan, misalnya untuk studi literatur.

Terhadap studi kasus ini, dimanfaatkan teknologi Oracle Text dalam hal penyimpanan dokumen terformat. Teknologi ini bertujuan memudahkan manajemen data-data dokumen yang berformat, seperti Microsoft Word, Microsoft Power Point, PDF, dan lainnya. Dengan Oracle Text, dapat pula dilakukan pencarian pada dokumen-dokumen tersebut. Sehingga pada tugas akhir

lunak. Untuk tujuan inilah teknologi XML dikembangkan. Sehingga nantinya berbagai perangkat lunak yang dihasilkan oleh berbagai vendor dapat berkomunikasi, bertukar data melalui sebuah format data, dan dapat memproses data tersebut. Dalam format XML, data dapat disimpan dalam format yang netral dari sisi platform, dan juga netral dari sisi bahasa pemrograman. Dengan *Document Type Definition* yang merupakan salah satu bagian dari spesifikasi XML, format data XML dapat dikembangkan sesuai dengan kebutuhan tanpa kehilangan konsistensi format data.

Kombinasi teknologi JSF dalam pengembangan perangkat lunak dengan Model 2X diharapkan dapat menghasilkan sebuah perangkat lunak yang memiliki layer presentasi yang lebih modular dan fleksibel dibandingkan dengan Model 1 atau JSP pada umumnya. Sebagai studi kasus, dibuat sebuah aplikasi Pengarsipan Tugas Akhir. Adapun pertimbangan penulis memilih studi kasus ini karena adanya kebutuhan akan perangkat lunak yang mengatur pengarsipan tugas akhir hasil karya mahasiswa. Sehingga tugas akhir yang pernah dibuat oleh mahasiswa tersimpan dengan baik dan dapat ditelusuri kembali dengan cepat ketika dibutuhkan, misalnya untuk studi literatur.

Terhadap studi kasus ini, dimanfaatkan teknologi Oracle Text dalam hal penyimpanan dokumen terformat. Teknologi ini bertujuan memudahkan manajemen data-data dokumen yang berformat, seperti Microsoft Word, Microsoft Power Point, PDF, dan lainnya. Dengan Oracle Text, dapat pula dilakukan pencarian pada dokumen-dokumen tersebut. Sehingga pada tugas akhir

ini selain menerapkan arsitektur aplikasi yang fleksibel, juga memanfaatkan teknologi penyimpanan dokumen pada database Oracle.

1.2. PERMASALAHAN

Permasalahan yang diangkat dalam tugas akhir ini adalah:

1. Implementasikan arsitektur Model 2X dalam studi kasus Situs Pengarsipan Tugas Akhir.
2. Integrasi teknologi JavaServer Faces ke dalam sebuah aplikasi berbasis web dengan arsitektur Model 2X.
3. Mengimplementasikan penanganan antarmuka yang fleksibel sebagai keunggulan dari Model 2X
4. Mengimplementasikan teknologi Oracle Text untuk penyimpanan dokumen dan pencarian dokumen.

1.3. BATASAN MASALAH

Dari permasalahan diatas, maka batasan masalah dalam tugas akhir ini ialah:

- Data yang disimpan di Situs Pengarsipan Tugas Akhir dikhususkan pada dokumen-dokumen yang menyertai Tugas Akhir.
- Fitur pokok yang akan dimiliki oleh Situs Pengarsipan Tugas Akhir ini adalah administrasi data tugas akhir yang meliputi menambah dan mengupdate data-data tugas akhir. Sedang fitur lain adalah aplikasi ini juga menyimpan artikel ilmiah/tulisan yang dapat dijadikan sebagai referensi tugas akhir.
- Format keluaran dari arsitektur Model 2X dibatasi pada format *HyperText Markup Language* (HTML).

1.4. TUJUAN

Tujuan dari pembuatan tugas akhir ini ialah untuk menerapkan arsitektur Model 2X ke dalam bentuk nyata berupa sebuah aplikasi berbasis web yang sekaligus menerapkan teknologi JavaServer Faces dan XML/XSLT. Sebagai studi kasus, penulis memilih pembuatan Situs Pengarsipan Tugas Akhir. Dari aplikasi ini diharapkan dapat memberikan manfaat:

1. Memperlihatkan bentuk nyata sebuah aplikasi berbasis web yang menerapkan arsitektur Model 2X.
2. Memperlihatkan bentuk implementasi teknologi JSF. Dari aplikasi ini akan diperlihatkan kelebihan-kelebihan dari teknologi JSF.
3. Mempermudah pencarian pada data berupa dokumen terformat dengan memanfaatkan teknologi Oracle Text.

1.5. METODOLOGI PEMBUATAN TUGAS AKHIR

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.5.1. Studi Literatur

Studi literatur dilakukan dilakukan untuk mempelajari konsep dari teknologi yang digunakan. Informasi diperoleh baik melalui buku maupun referensi dari internet. Tahap ini merupakan tahap pengumpulan informasi yang diperlukan untuk memahami lebih mendalam tentang arsitektur Model 2X, Teknologi JavaServer Faces, XML/XSLT serta Oracle Text.

1.5.2. Perancangan Sistem dan Aplikasi

Pada tahap ini ditentukan bagaimana struktur dan komponen-komponen Aplikasi web dalam memenuhi bentuk arsitektur Model 2X. Untuk membangun Aplikasi yang dijadikan Studi kasus, ditentukan juga kelas-kelas yang akan merepresentasikan model objek pada sistem. Dalam tahapan ini digunakan notasi Unified Modeling Language (UML) untuk membantu menggambarkan desain untuk proses perancangannya.

1.5.3. Pembuatan Aplikasi

Dalam tahap ini, dilakukan implementasi berdasarkan desain perangkat lunak yang telah dibuat pada tahap sebelumnya dengan menggunakan teknologi-teknologi yang telah didapatkan melalui metodologi nomor satu di atas.

1.5.4. Uji Coba dan Evaluasi

Pada tahap ini dilakukan uji coba perangkat lunak untuk memperlihatkan integrasi antara teknologi JavaServer Faces dan XML/XSLT sebagai bentuk arsitektur Model 2X. Juga dalam hal pemanfaatan teknologi Oracle Text untuk pencarian dalam dokumen. Beberapa skenario akan digunakan agar dapat dilakukan evaluasi apakah tujuan-tujuan yang telah dirumuskan telah tercapai.

1.5.5. Penyusunan Buku Tugas Akhir

Setelah kesemua tahap terdahulu terselesaikan, disusun buku Tugas Akhir sebagai dokumentasi dari pelaksanaan tugas akhir.

1.6 SISTEMATIKA PEMBAHASAN TUGAS AKHIR

Buku tugas akhir ini terdiri dari abstrak yang membahas mengenai ringkasan dari tugas akhir secara umum, kata pengantar, daftar isi, daftar tabel dan daftar gambar. Kemudian isi dari buku tugas akkhir ini terdiri dari beberapa bab, yang masing-masing dijelaskan sebagai berikut:

1. **BAB I Pendahuluan**

   Bab ini menjelaskan beberapa hal pokok dari tugas akhir ini, antara lain : latar belakang,yang mendasari pembuatan tugas akhir, perumusan masalah, batasan masalah, tujuan pembuatan tugas akhir, metodologi yang berkaitan dengan pengerjaan tugas akhir ini dan sistematika penulisan tugas akhir ini.

2. **BAB II Dasar Teori**

   Pada bab ini dibahas secara singkat teori-teori yang digunakan sebagai referensi dalam pengerjaan tugas akhir ini, meliputi: Teknologi XML, XSL, JSP dan JSF, penjelasan mengenai Oracle Text sebagai fitur dari Oracle untuk penyimpanan dokumen, serta penjelasan mengenai arsitektur Model 2X

3. **BAB III Perancangan Sistem**

   Pada bab ini dijabarkan mengenai tahapan-tahapan dari proses perancangan dan pembuatan dari tugas akhir ini, yang meliputi tiga perancangan, yaitu: perancangan data, perancangan proses dan perancangan antarmuka. Perancangan data menjelaskan rancangan basis data yang digunakan. Perancangan proses dijelaskan dalam bentuk diagram UML. Dan perancangan antarmuka menjelaskan antarmuka yang dibuat.

4. **BAB IV Implementasi Perangkat Lunak**

   Bab ini membahas bagaimana aplikasi dibuat berdasarkan rancangan-rancangan yang telah dibuat pada bab sebelumnya. Bab ini dibagi atas: implementasi data, implementasi proses, serta implementasi antarmuka.

5. **BAB V Uji coba dan Evaluasi Aplikasi**

   Bab ini digunakan untuk membahas pengujian dan evaluasi. Hasil dari pengembangan aplikasi akan diuji dan dievaluasi dengan berbagai macam kondisi sehingga tercapai hasil yang sesuai dengan yang diinginkan.

6. **BAB VI Kesimpulan dan Saran**

   Bab ini berisi kesimpulan–kesimpulan yang dapat diambil dari proses pengembangan aplikasi tugas akhir ini. Juga berisi saran-saran untuk kepentingan pengembangan selanjutnya.

BAB II

DASAR TEORI

BAB II

DASAR TEORI

Pada bab ini diberikan penjelasan dasar teori dari teknologi-teknologi yang mendasari pembuatan tugas akhir ini. Bab II ini dibagi menjadi empat sub bab, masing-masing membahas teknologi XML, JavaServer Faces, Oracle Text, serta teknologi arsitektur Model 2X.

2.1. TEKNOLOGI XML

Sub bab ini memberi penjelasan teknologi XML yang digunakan pada tugas akhir ini. Nantinya teknologi XML ini digunakan ketika membangun *presentation layer* pada aplikasi pengarsipan tugas akhir.

XML merupakan salah satu markup language, yaitu bahasa pemrograman untuk menandai suatu data. *Extensible Markup Language*, disingkat XML, menerangkan sebuah kelas obyek data yang dinamakan dokumen XML dan menerangkan sebagian tindakan yang dilakukan program komputer untuk memprosesnya.

Ada dua hal penting agar sebuah markup dapat memberikan informasi penting dalam suatu data, yaitu :

1. Harus ada standar yang menjelaskan suatu markup itu adalah valid.
2. Harus ada standar yang menjelaskan apa arti markup tersebut.

Dalam aplikasinya, XML sendiri adalah markup language yang menyediakan format untuk mendeskripsikan data terstruktur. Contoh data

terstruktur adalah dokumen *spreadsheet*, buku alamat, parameter konfigurasi, transaksi keuangan, gambar teknis dll. Pada XML kita bisa mendefinisikan kumpulan tag yang tak terbatas. XML merupakan suatu standar jika memenuhi suatu DTD atau *schema* yang telah disepakati, maka seseorang dapat memanipulasi data XML tersebut, tanpa peduli aplikasi yang menggunakan XML tersebut. XML sangat erat terikat dengan internet karena beberapa alasan, salah satunya adalah karena dokumen XML hanyalah sebuah dokumen text sederhana yang sangat mudah melewati protokol internet, sistem operasi dan *firewall*.

XML sebagai format yang bisa didefinisikan sendiri mempunyai beberapa keuntungan dan keunggulan :

- Ekstensibilitas, artinya bebas menentukan tag-tag sendiri sesuai dengan kebutuhan.
- Pemisahan data dengan presentasi, artinya sebuah dokumen XML itu hanya berisi data saja yang nanti akan ditampilkan oleh layer presentasi.
- Mudah untuk melakukan parsing terhadapnya, karena terformat dalam tag-tag dalam file plain text.
- Bisa mempunyai schema eksternal (DTD, *XML schema*) sebagai meta-model yang memvalidasi XML.
- Independen terhadap platform, vendor dan aplikasi, artinya tidak bergantung pada platform, vendor dan aplikasi dimana XML digunakan.

2.1.1. Spesifikasi Teknis XML

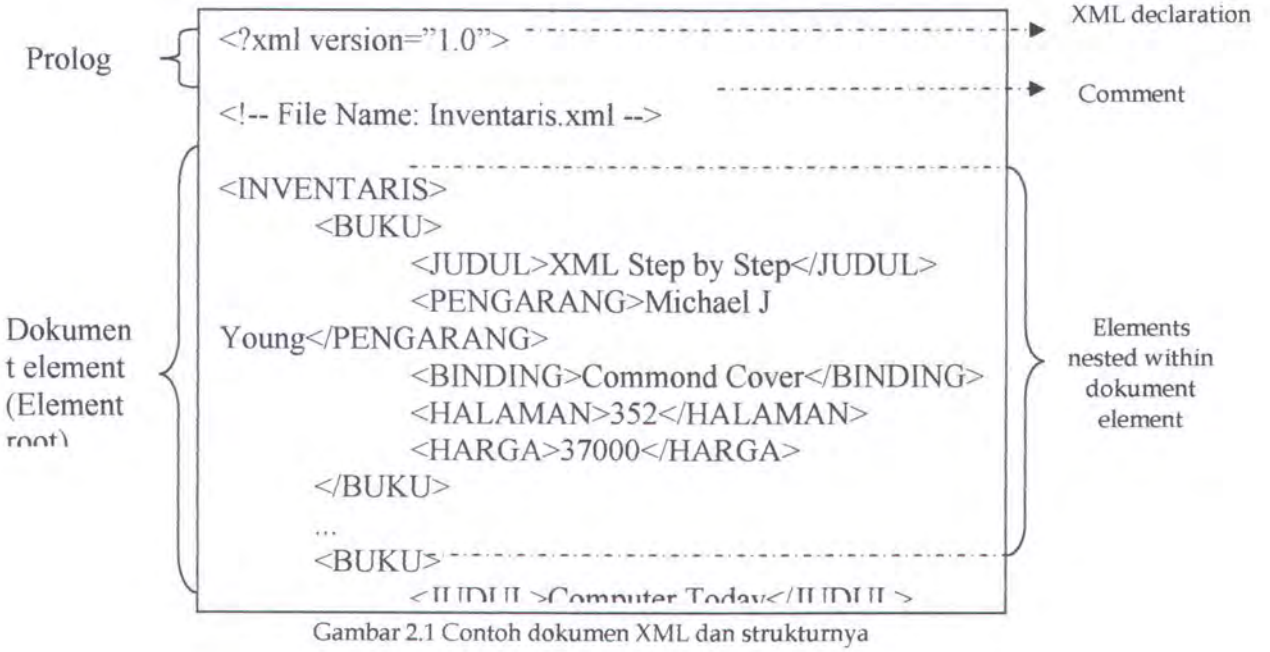
2.1.1.1. Anatomi Dokumen XML

Sebuah dokumen XML terdiri dari dua bagian utama : prolog dan elemen dokumen *(element root)*.

- Prolog

Deklarasi XML menyatakan bahwa file tersebut adalah sebuah dokumen XML dan menampilkan nomor versinya. Deklarasi XML bersifat opsional, walaupun spesifikasinya menyatakan bahwa ia harus dimasukkan, dan letakknya harus di awal dokumen.

Baris ketiga prolog adalah sebuah komentar. Penambahan komentar ke sebuah dokumen XML bersifat opsional. Sebuah komentar diawali dengan karakter <!-- dan diakhiri dengan karakter -->, sembarang teks akan diabaikan jika ditulis diantara karakter tersebut.



Gambar 2.1 Contoh dokumen XML dan strukturnya

- Elemen Dokumen

Bagian utama kedua adalah sebuah elemen tunggal yang disebut dengan elemen dokumen yang disebut elemen dokumen atau elemen root, yang bisa berisi elemen-elemen tambahan.

Elemen-elemen menandakan struktur logika sebuah dokumen dan berisi isi informasi dokumen. Kebanyakan elemen berisi sebuah tag-awal, isi elemen, dan sebuah tag akhir. Isi elemen bisa berupa data karakter, elemen (tersarang) lain, atau kombinasi keduanya.

2.1.1.2. Sejumlah Aturan Dasar XML

Dokumen yang *well-formed* adalah dokumen yang tunduk pada seperangkat aturan minimal yang memungkinkan dokumen diproses oleh browser atau program lainnya. Berikut adalah sebagian aturan dasar untuk pembuatan dokumen XML yang rapi atau *well-formed.*

- Dokumen harus mempunyai satu elemen tingkat teratas (*element root*).
- Elemen-elemen harus disarangkan dengan tepat. Yakni, jika sebuah elemen diawali dalam elemen lain, ia harus juga berakhir dalam elemen yang sama.
- Setiap elemen harus memiliki tag-awal dan tag-akhir. XML tidak mengijinkan untuk mengabaikan tag-akhir.
- Nama tipe-elemen dalam tag-awal harus persis berhubungan dengan nama dalam tag-akhir yang bersangkutan.

2.1.2. XSL (eXtensible Stylesheet Language)

XSL atau *Extensible Stylesheet Language* adalah sebuah bahasa untuk mengekspresikan *style sheet.* Sebuah XSL dikaitkan pada sebuah dokumen XML

dan memberitahu browser bagaimana menampilkan data XML, juga memungkinkan untuk membuka dokumen XML langsung dalam browser tanpa menggunakan halaman perantara.

XSL memungkinkan pemilihan dengan tepat data XML yang ingin ditampilkan untuk kemudian disajikan dalam berbagai susunan atau tatanan, dan untuk bebas mengubah dan menambahkan informasi. XSL memberikan akses ke semua komponen XML (seperti elemen, atribut, komentar dan instruksi pemrosesan) memungkinkan proses menyortir dan mengurutkan data XML.

Karena XML tidak menggunakan tag-tag yang sudah didefinisikan seperti HTML (kita bisa menggunakan tag yang kita inginkan), maka arti dari tag-tag XML tersebut tidak bisa dimengerti secara pasti. Tag `<table>` bisa berarti table dalam HTML, atau sebuah meja, atau sesuatu yang lain. Sebuah *browser* tidak tahu bagaimana untuk menampilkan sebuah dokumen XML. Oleh karenanya harus ada sesuatu sebagai tambahan dalam dokumen XML yang mendeskripsikan bagaimana dokumen harus ditampilkan, dan itu adalah XSL. XSL terdiri dari tiga bagian yaitu :

- XSLT, sebuah bahasa untuk mentransformasi dokumen XML.
- XPath, sebuah bahasa untuk menyebut bagian dari sebuah dokumen XML.
- XSL Formating Object, sebuah kosakata untuk memformat dokumen XML.

XSL adalah sebuah bahasa yang bisa melakukan **transformasi** XML menjadi XHTML, sebuah bahasa yang bisa melakukan **filter** dan **pengurutan** data XML, sebuah bahasa yang bisa **mendefinisikan bagian** dari dokumen XML, sebuah bahasa yang bisa **memformat** data XML berdasarkan nilai data dan

sebuah bahasa yang bisa **mengeluarkan** data XML ke device/alat yang berbeda seperti layar monitor, kertas atau suara.

XSL adalah standar yang direkomendasikan oleh konsorsium World Wide Web, dimana dua bagian pertama dari XSL diatas menjadi rekomendasi W3C pada November 1999.

2.1.2.1. XSL Transformation (XSLT)

XSLT adalah bagian terpenting dari standar XSL. XSLT adalah bagian dari XSL yang digunakan untuk mengubah sebuah dokumen XML menjadi dokumen XML yang lain, atau tipe dokumen yang lain yang dikenal oleh browser. Salah satu format dokumen tersebut adalah XHTML. Secara normal XSLT melakukan transformasi dengan mengubah tiap elemen XML menjadi sebuah elemen XHTML.

XSLT bisa juga menambah elemen baru kedalam file luaran, atau menghapus elemen. Ia bisa menyaring dan mengurutkan elemen, melakukan test dan membuat keputusan tentang elemen mana yang akan ditampilkan, dan banyak lagi.

Dalam proses transformasi, XSLT menggunakan XPath untuk mendefinisikan bagian dari dokumen sumber yang cocok dengan satu atau lebih template yang telah didefinisikan. Ketika ada kecocokan, XSLT akan mentransform bagian yang cocok dari dokumen sumber kedalam halaman hasil. Bagian dari dokumen sumber yang tidak cocok dengan template tidak akan di modifikasi pada halaman hasil.

2.1.2.2. Contoh Penggunaan XSL

Elemen root yang mendeklarasikan dokumen menjadi sebuah *XSL Style Sheet* adalah `<xsl:sytlesheet>` atau `<xsl:transform>`, dimana keduanya adalah sinonim dan semuanya bisa digunakan. Cara yang benar untuk mendeklarasikan sebuah *XSL stylesheet* berdasarkan rekomendasi XSL W3C adalah :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="/">
  <html>
  <body>
    <h2>Koleksi Buku</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
      <th align="center">Judul</th>
      <th align="center">Pengarang</th>
      <th align="center">Binding</th>
      <th align="center">Halaman</th>
      <th align="center">Harga</th>
    </tr>
    <xsl:for-each select="INVENTARIS/BUKU">
    <tr>
      <td><xsl:value-of select="JUDUL"/></td>
      <td><xsl:value-of select="PENGARANG"/></td>
      <td><xsl:value-of select="BINDING"/></td>
      <td><xsl:value-of select="HALAMAN"/></td>
      <td><xsl:value-of select="HARGA"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Gambar 2.2 Contoh dokumen XSL, inventaris.xsl

Contoh dokumen XML di bagian sebelumnya (`inventaris.xml`) akan di transform kedalam bentuk XHTML. Sebelumnya harus dibuat *XSL stylesheet*-nya terlebih dahulu dengan nama file inventaris.xml, seperti pada contoh berikut.

```
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="inventaris.xsl"?>

<INVENTARIS>
      <BUKU>
            <JUDUL>XML Step by Step</JUDUL>
            <PENGARANG>Michael J Young</PENGARANG>
            ...
```

Gambar 2.3 Penggunaan XSL pada dokumen XML

Agar dokumen XML (`inventaris.xml`) bisa ditampilkan dengan XSL (`inventaris.xsl`), maka pada dokkumen XML perlu ditambahkan sebuah referensi yang menunjuk ke dokumen XSL yang akan dipakai.

2.1.2.3 Menyaring dan Menyortir data XML

- Penyaringan

Seperti contoh penggunaan XSL diatas, nilai yang diberikan ke atribut `match` atau `select` adalah pola yang berhubungan dengan satu atau lebih elemen dalam dokumen XML. (atribut `match` dengan elemen `template`, dan atribut `select` dengan elemen `value-of`, `for-each`, dan `apply-template`). Pola-pola tersebut sejauh ini hanya berisi operator path yang menyebutkan nama elemen, dan mungkin pula satu atau lebih elemen pendahulu. Kita bisa membatasi lebih jauh jumlah elemen yang sesuai pola dengan memasukkan sebuah filter dikelilingi oleh kurung persegi ([]) tepat setelah operator path. Misalnya, pola yang diberikan ke atribut match berikut menunjukkan bahwa elemen yang bersangkutan harus memiliki nama `BUKU` dan selanjutnya (bagian filter) harus memiliki sebuah elemen `JUDUL` anak yang berisi teks "Computer Today".

```
<xsl:template match="BOOK[JUDUL='computer today']">
```

Gambar 2.4 Contoh *filtering* data XML menggunakan XSL

Cukup memasukkan sebuah nama elemen dalam filter tersebut menandakan bahwa elemen yang bersangkutan harus memiliki sebuah elemen anak dengan nama yang dimasukkan.

- Penyortiran

Kita bisa menggunakan atribut `order-by` untuk mengontrol urutan di mana browser memproses elemen tersebut, sehingga menyortir data XML yang ditampilkan.

Atribut `order-by` bisa diberika dalam satu atau beberapa pola, dipisahkan dengan semicolon. Browser akan menyortir elemen dengan pola dalam urutan penyantuman mereka. Untuk menunjukkan susunan menurun atau naik, setiap pola didahului dengan tanda + atau -.

Sebagai contoh, seting atribut `order-by` dalam elemen `for-each` berikut menyebabkan browser mengurutkan elemen `BUKU` dalam urutan naik (ascending) berdasar nama terakhir `PENGARANG`.

```
<xsl:for-each select="INVENTARI/BUKU" order-by="+PENGARANG">
```

Gambar 2.5 Contoh *sorting* data XML menggunakan XSL

Operator path yang diberikan pada atribut `order-by` menyangkut pola yang diberikan pada atribut `select`. Dengan demikian, dalam contoh ini, seting `order-by="+PENGARANG"` menunjukkan elemen `PENGARANG` dalam elemen `BUKU`, dalam elemen `INVENTARIS`.

2.2 JSF (Java Server Faces)

Pada bagian ini akan dijelaskan mengenai teknologi JavaServer Faces berikut komponen-komponen yang menyusun JSF. Akan dijelaskan terlebih dahulu komponen-komponen yang menyusun JSF, yaitu: Servlet, Servlet Filter, dan JSP. Kemudian dilanjutkan dengan penjelasan JSF pada bab 2.2.3.

2.2.1. Servlet dan Servlet Filter

2.2.1.1. Servlet

Beberapa tahun yang lalu, metode paling populer untuk membuat aplikasi web dinamis adalah dengan Common Gateway Interface (CGI). CGI adalah gateway(penghubung) antara antara web server dan kode CGI, program CGI dapat mengurangi kemampuan sistem jika volume web yang tinggi karena web server harus melakukan proses yang baru setiap waktu ketika kode CGI berjalan. Pada tahun 1997, Sun Microsystems mengenalkan standar *Application Programming Interface* (API) yang dinamakan Java Servlet untuk mengembangkan komponen Java Server-side. Keuntungan paling utama dari servlet disbanding kode CGI adalah servlet dapat dijalankan satu kali untuk melayani beberapa client. Termasuk request yang baru, sebuah servlet tidak perlu menyediakan proses baru yang mengakibatkan prosesor semakin lambat.

Servlet mempunyai beberapa keuntungan dibandingkan dengan CGI :

- Servlet disimpan di memori dan tidak menyediakan proses baru, sehingga tidak ada penambahan proses setiap ada permintaan proses baru.
- Sebuah servlet dapat menjawab semua request secara bersamaan.

- Servlet berjalan di servlet container yang dapat mengamankan dari servlet yang mempunyai potensi merusak
- Servlet terintegrasi penuh dengan framework J2EE
- Java servlet didukung oleh hamper semua web server seperti Netscape,Sun, IBM dan Apache. Karena servlet ditulis dalam bahasa java, servlet dapat dipakai disemua platform yang mempunya Java Virtual Machime (JVM) dan web server yang mendukung servlet.

Konsep dasar Servlet

Teknologi servlet menyediakan mekanisme untuk mengembangkan fungsi dari web server dan untuk mengakses proses bisnis. Servlet mempunyai akses ke semua produk java API termasuk JDBC API untuk mengakses basis data. Servlet juga dapat mengakses *library* HyperText Transfer Protocol (HTTP) dan menerima semua keuntungan dari bahasa Java, termasuk kemudahan, performance, *reusability* dan kemanan.

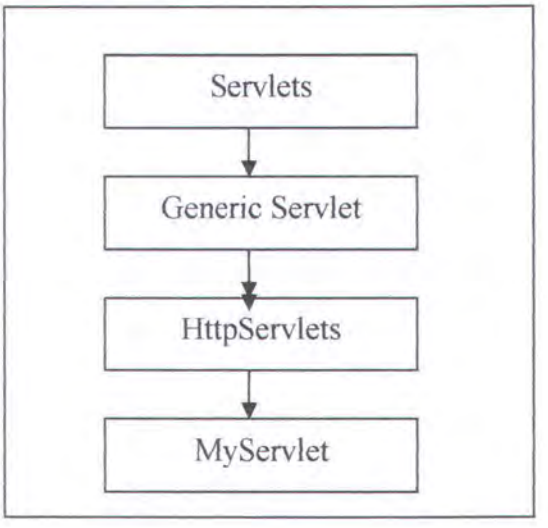
Cara Kerja Protokol HTTP

HTTP berorientasi pada respon terhadap sebuah permintaan. Sebuah HTTP request terdiri dari metode *request*, sebuah Uniform Resource Identifier (URI), *field header* dan sebuah *body*. Web *client* menggunakan HTTP untuk berkomunikasi dengan web server. HTTP mendefinisikan permintaan sehingga *client* dapat mengirim ke server dan merespon yang menyebabkan server dapat menjawab balik. Setiap *request* terdiri dari Uniform Resource Locator (URL), yaitu sebuah string yang mengidentifikasi komponen web atau sebuah obyek seperti halaman HTML dan file gambar.

Server web mengubah request HTTP ke obyek HTTP dan mengirim ke komponen web yang diidentifikasi oleh URL. HTTP response terdiri dari sebuah hasil kode seperti field header dan sebuah body.

Arsitektur Dasar Servlet

Sebuah servlet adalah sebuah kelas java yang mengimplementasikan *interface* `javax.servlet.Servlet`.



Gambar 2.6 Standard servlet interface

Ada dua tipe utama servlet :

- **Generic servlets**. Meliputi `javax.Servlet.GenericServlets`. Generic Servlets tidak hanya mendukung HTTP, tetapi juga jenis protokol yang lain.
- **HTTP servlets.** Meliputi `javax.servlet.HttpServlet`. HTTP servlets dibuat khusus untuk mendukung protokol HTTP dan digunakan di lingkungan browser internet.

Kedua tipe servlet menggunakan metode konstruktor `init()` untuk menginisialisasi sumber (*resource*) dan metode destruktor `destroy()` untuk

melepas *resource* yang tidak dibutuhkan lagi. Semua tipe servlet harus mengimplementasikan metode `service()` yang mengatur *request* ke sevlet.

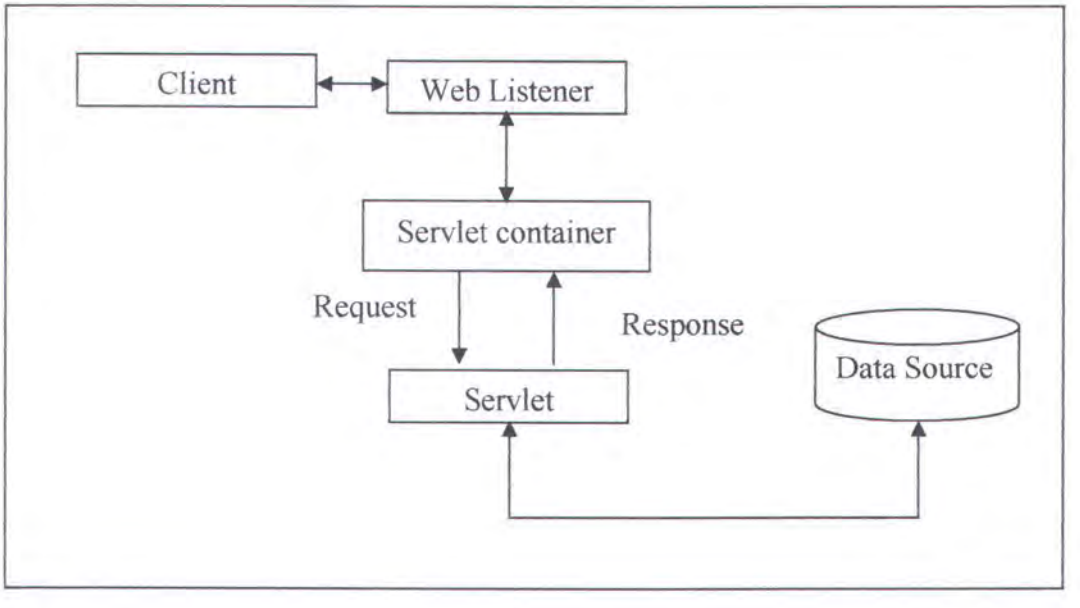
Berikut ini adalah metode servlet untuk menangani request:

- `init(ServletConfig config)`
- `service(ServletRequest req, ServletResponse res)`
- `destroy( )`

Servlet Containers

Tidak seperti program aplikasi client java, sevlet tidak mempunyai fungsi `main()`. Servlet harus dieksekusi dibawah kontrol servlet *container*, karena container yang membuat dan menghapus servlet instance, memanggil metode servlet dan menyediakan layanan servlet yang dibutuhkan saat proses eksekusi.

Container servlet menyediakan kemudahan akses ke HTTP *request* seperti header dan parameter. Servlet dapat menggunakan Java API seperti JDBC atau SQLJ untuk mengakses database, Remote Method Invocation (RMI) untuk memanggil obyek.



Gambar 2.7 Interaksi antar servlet container

Berikut ini adalah daur hidup dari sebuah Servlet:

1. *Container* membuat *instance* baru java servlet ketika Servlet *Container* mulai berjalan.
2. Ketika java servlet *loading*. Konfigurasi secara detail dibaca dari web.xml termasuk inisialisasi parameter.
3. Hanya terdapat satu *instance* java servlet. Client meminta secara bersama-sama java servlet instance.
4. Client memanggil fungsi `service( )` dari generic servlet, dan melanjutkan ke metode `doGet( )` atau `doPost( )`, tergantung informasi di *header*
5. Java servlet dapat meneruskan *request* ke java servlet yang lain.
6. Java servlet membuat objek *response*.
7. *Container* memanggil metode `destroy( )` sebelum menghentikan proses servlet.

2.2.1.2. Filter dan Chaining

Ketika servlet container memanggil metode servlet sesuai permintaan client yaitu HTTP *request*. Objek *request* dari *client* secara langsung diarahkan ke servlet. dan *response* dari servlet secara langsung dikembalikan ke *client* dengan isi informasi yang tidak dimodifikasi oleh *container*. Jadi secara normal, servlet harus memproses *request* dan membangkitkan banyak *response* sebagai kebutuhan aplikasi.

Tetapi dalam beberapa kasus, proses-proses terhadap *request* sebelum *request* masuk ke servlet akan sangat berguna. Salah satu bentuknya adalah untuk memodifikasi *response* dari kelas servlet. Sebagai contoh adalah proses enkripsi.

Servlet dapat membangkitkan *response* data yang sensitif dan tidak boleh keluar jaringan, khususnya ketika koneksi telah membuat protokol yang kurang aman seperti HTTP. Filter dapat melakukan enkripsi *response*.

Pada kasus umum filter digunakan pada saat melakukan awalan proses dan sesudah proses untuk *request* dan memberi *response* kumpulan servlet dibanding satu servlet. Jika dengan satu servlet, tidak dibutuhkan sebuah filter, langsung menjalankan sesuai permintaan ke servlet itu sendiri.

Perlu diperhatikan bahwa filter bukan sebuah servlet. Filter mengimplementasikan metode `javax.servlet.Filter` interface. Filter API didefinisikan pada *interface* `Filter`, `FilterChain` dan `FilterConfig` di paket `java.servlet`.

Beberapa metode dari FilterChain :

- `init( )`

  Melakukan inisialisasi terhadap filter. Disini dapat dilakukan pembacaan konfigurasi yang terdapat pada file deskriptor `web.xml`

- `destroy( )`

  Dipanggil ketika Filter akan diambil oleh *garbage collector*. Disini dilepaskan resource yang tidak dipakai lagi.

- `doFilter( )`

  Operasi-operasi yang dilakukan terhadap Objek *request* dan *response* dikerjakan di sini. Method ini merupakan method yang paling penting dibandingkan dengan yang lain, karena di sini terdapat inti dari proses Filter itu sendiri.

2.2.2. JSP (JavaServer Pages)

JavaServer Pages adalah spesifikasi yang dikeluarkan oleh Sun Microsystem untuk membuat aplikasi berbasis web yang dinamis. Dengan spesifikasi ini kode java dapat diselipkan di antara kode-kode HTML. Cara ini jauh lebih mudah daripada penggunaan HttpServlet yang mengharuskan kode java menghasilkan kode HTML. Isi sebuah file JSP dapat dibagi menjadi dua kategori utama, yaitu:

- Elemen, sesuatu yang akan diproses di sisi server
- Template data, atau segala sesuatu selain elemen, yang tidak diproses oleh server, melainkan langsung ditampilkan.

Elemen dari file JSP dapat berupa:

- JSP Directives
- JSP Declarations
- Scriplet
- JSP Expressions
- Standard Actions

2.2.2.1. JSP Directives

JSP *Directives* berperan sebagai pesan khusus bagi JSP *Container* yang diberikan oleh file JSP. JSP *Directives* digunakan untuk menentukan setting global seperti deklarasi kelas, *Content-type*, halaman error, dan lainnya. JSP *Directives* tidak menghasilkan luaran apapun bagi client. Lingkup yang dimiliki oleh JSP *Directive* adalah lingkup *page*. JSP *Directives* ditandai dengan karakter "@" pada tag-nya. Sintaksnya adalah:

```
<%@directive attribute="value"%>
```

Gambar 2.8 Sintaks JSP Directives

Dimana directive dapat berupa:

1. `page`: Directive `page` digunakan untuk memberikan informasi tentang page tersebut.

   Pada Directive `page`, atribut dapat berupa:

   - `language="java"`. Memberi informasi bahasa pemrograman yang digunakan. Spesifikasi JSP saat ini hanya dapat mendukung bahasa pemrograman java.
   - `import="mypackage.myclass"`. Atribut ini berguna untuk melakukan import kelas atau package tertentu seperti halnya pada sebuah program java.
   - session="true". Jika nilai ini diset true, maka data pada session dapat diakses pada halaman ini, nilai default Atribut ini adalah true.
   - `errorPage="error.jsp"`. Atribut ini berguna untuk mengarahkan alur program ke halaman khusus jika terjadi error.
   - `contentType="text/html;charset=ISO-8859-1"`. Atribut ini digunakan untuk menetukan mime type dan character set dari file JSP.

2. `include`. Directive ini Digunakan untuk menyisipkan file lain pada file JSP.
3. `taglib`. Directive ini digunakan ketika akan memakai *costum* tag pada file JSP. Dengan *costum* tag pengembang dapat mendefinisikan tag-nya sendiri sesuai kebutuhan.

Berikut contoh penggunaan ketiga *directive* yang telah dijelaskan di atas.

```
<%page language="java" import="java.sql.*,mypackage.class"
      session="true" errorPage="error.jsp"
      contentType="text/html;charset=ISO-8859-1"%>
<%@include file="/header.jsp"%>
<%@taglib uri="tlds/taglib.tld" prefix="mytag"%>
```

Gambar 2.9 Contoh Directive Page, Include, dan Taglib.

2.2.2.2. JSP Declarations

Elemen JSP *Declaration* ditandai dengan tag <%! dan tag %>. Elemen ini berfungsi sebagai tempat deklarasi variable dan fungsi. Variabel dan fungsi yang dideklarasikan di sini memiliki *scope* terbatas pada *page*. Berikut diberikan contoh penggunaan Elemen JSP *Declaration.*

```
<%!
int cnt=0;
private int getCount(){
//increment cnt and return the value
cnt++;
return cnt;
}
```

Gambar 2.10 Contoh penggunaan JSP Declaration

2.2.2.3. Scriplet

Scriplet adalah sekumpulan kode java yang dilingkupi oleh tag scriplet dan dieksekusi oleh JSP *Container* pada proses request dari client. Scriplet dilingkupi dengan tag <% dan %>. Pada file JSP sejumlah scriplet dapat diselipkan di antara kode-kode HTML. Ketika diproses oleh JSP *Container*, Scriplet akan diubah menjadi kode java sesuai dengan urutan pada file JSP.

```
<%
  //java codes
   String userName=null;
   userName=request.getParameter("userName");
   %>
```

Gambar 2.11. Contoh scriplet pada file JSP

2.2.2.4 JSP Expressions

JSP *Expression* adalah bentuk ringkas dari scriplet yang mengevaluasi sejumlah kode java. Dari kode java ini dihasilkan luaran berupa String untuk ditampilkan ke *client*. JSP *Expression* ditandai dengan tag <%= dan %>. Berikut ini diberikan contoh JSP *Expression* beserta luaran HTML-nya.

```
<html>
<%! String name="Java Server Pages"%>
language name : <%= name %>
</html>
```

Gambar 2.12 Contoh JSP Expression

```
<html>
language name : Java Server Pages
</html>
```

Gambar 2.13 Luaran dari file JSP pada contoh 2.11.

2.2.2.5. Standard Actions

Yang dimaksud dengan *Actions* di sini adalah tag-tag khusus yang mempengaruhi alur eksekusi JSP dan juga mempengaruhi *response* yang akan dikirim kembali ke *client*. Berikut penjelasan beberapa *Standard Actions* pada JSP:

- `<jsp:useBean>`

  Action `<jsp:useBean>` digunakan untuk menghubungkan antara sebuah JavaBean dengan JSP. Dengan tag ini objek JavaBean dapat diakses dari JSP dengan id dan *Scope* yang sesuai. Tipe objek JavaBean yang diakses, id dan *scope* dari JavaBean itu ditentukan pada tag `useBean`.

- `<jsp:setProperty>`

  Action ini digunakan untuk mengisi nilai *property* dari JavaBean yang telah dihubungkan dengan tag useBean.

- `<jsp:getProperty>`

  Action ini digunakan untuk mengambil nilai *property* dari JavaBean yang telah dihubungkan dengan tag `useBean`. Nilai dari *property* yang diambil langsung diubah menjadi String dan ditampilkan melalui *OutputStream*.

- `<jsp:param>`

  Action `<jsp:param>` digunakan untuk memberikan informasi tambahan berupa pasangan nama dan nilai yang berfungsi sebagai parameter. Informasi ini dibutuhkan oleh 3 (tiga) tag yang akan dijelaskan pada poin yang berikut.

- `<jsp:include>`

  Action ini akan menyisipkan masukan statis ataupun dinamis pada file JSP ketika JSP *container* memproses request yang masuk. Masukan disini dapat berupa file JSP, file HTML ataupun gambar. Masukan yang disisipkan ditentukan dalam format URL. Tag ini dapat ditambahkan dengan tag `<jsp:param>` untuk memberikan parameter bagi masukan yang akan disisipkan.

- `<jsp:forward>`

  Action `<jsp:forward>` akan meneruskan *request* ke file JSP lain, atau Servlet ataupun hanya sebuah file statik seperti HTML. action ini hanya akan melakukan *forward* dalam *context* yang sama.

- <jsp:plugin>

  Action `<jsp:plugin>` action digunakan untuk menghasilkan tag HTML khusus pada *client* (OBJECT atau EMBED) serta memastikan bahwa perangkat lunak Java Plug-in tersedia, diikuti dengan eksekusi Applet ataupun JavaBean yang telah ditentukan pada tag ini

2.2.3. JSF (Java Server Faces)

Untuk membangun sebuah aplikasi berbasis web dengan bahasa pemrograman Java, pengembang dapat menggunakan JSP dan Servlet. Akan tetapi teknologi ini masih terlalu mendasar untuk pengembangan antarmuka yang lebih kompleks. Pengembang masih harus membuat sendiri elemen-elemen antarmuka seperti validasi masukan, navigasi halaman. Belum ada teknik yang standar yang mengatur implementasi elemen-elemen antarmuka ini. Dan tanpa standar yang lengkap dan jelas, akan sulit bagi vendor untuk membuat tool pengembangan yang memberikan produktivitas dan visualisasi ketika membangun aplikasi berbasis web dengan bahasa pemrograman java. Untuk permasalahan inilah teknologi Java Server Faces (JSF) dibangun sebagai solusi yang lebih baik dalam pengembangan aplikasi berbasis web dalam bahasa pemrograman java.

JSF adalah teknologi yang dikembangkan oleh Sun Microsystem yang bertujuan untuk membuat *framework* standar untuk pembuatan komponen antarmuka pada aplikasi berbasis web. JSF berjalan pada server web berbasis java dan melakukan proses render antarmuka pada *client* yang berupa browser internet. JSF memberikan *life cycle management* dari sebuah aplikasi web melalui sebuah *Controller Servlet*. Selain itu JSF menyediakan sejumlah *component model* beserta penanganan even dan proses rendernya.

JSF pada intinya terdiri dari dua komponen utama:

- Sejumlah *Application Programming Interface (API)* untuk membuat komponen Antarmuka dan mengatur *state* komponen tersebut, menangani even-even dan validasi masukan serta memberikan navigasi halaman web.

- Sejumlah Tag Library JSP untuk merepresentasikan komponen JSF di dalam halaman JSP.

2.2.3.1. Keunggulan Teknologi JavaServer Faces

Keunggulan Utama dari teknologi JSF adalah adanya pemisahan yang jelas antara proses bisnis dengan antarmuka aplikasi. Aplikasi web yang dibangun dengan JSP hanya memiliki pemisahan secara parsial. Sebagai contoh, JSP tidak dapat mengarahkan *request* HTTP ke penanganan even yang telah disediakan komponen. JSP juga tidak dapat mengatur komponen antarmuka dari sisi server sebagai sebuah objek. JavaServer Faces mampu memberikan kemudahan bagi pengembang untuk membangun aplikasi berbasis web dengan pemisahan proses bisnis dan penanganan antarmuka yang biasanya didapatkan pada arsitektur antarmuka *client*.

Dengan pemisahan yang jelas ini, pengerjaan aplikasi web dapat dilakukan secara paralel. Jika pengembang berupa tim, anggota tim dapat memfokuskan pengerjaan pada bagiannya masing-masing. Misalnya *page Author* dapat bekerja membuat desain tampilan tanpa harus memiliki kemampuan pemrograman java. Sedangkan *Application Developer* dapat memfokuskan pekerjaannya pada proses bisnis aplikasi tanpa harus terganggu dengan desain antarmuka nantinya.

Selain itu JavaServer Faces juga menangani *state* dari komponen, memproses data komponen, melakukan validasi masukan dari pengguna, dan menyediakan penanganan even.

2.2.3.2. Framework Roles

Dengan adanya pemisahan pengerjaan, JavaServer Faces memberikan pembagian peran-peran dalam tim pengembang sebagai berikut:

- *Page Author*, yaitu orang yang menyusun desain antarmuka aplikasi web nantinya. Orang ini akan bekerja dengan bahasa HTML untuk menyusun prototipe aplikasi web.
- *Application Developer*, yaitu orang yang menyusun program untuk pemodelan objek, penanganan even, validasi, serta navigasi halaman. Ia juga dapat membuat *Helper Classes* untuk memudahkan pekerjaannya
- *Component Writers*, yaitu orang yang membuat komponen-komponen antarmuka khusus. Ia dapat membuat sendiri kelas-kelas komponennya atau dengan menurunkan dari kelas komponen yang standar.
- *Tool Vendor*, yaitu pihak yang menyediakan *tool* untuk memanfaatkan fitur-fitur JavaServer Faces secara lebih mudah.

2.2.3.3. Model Komponen Antarmuka

JavaServer Faces menyediakan sejumlah kelas komponen Antarmuka yang memiliki semua atribut-atribut komponen antarmuka. Contohnya menyimpan state komponen, menyimpan referensi ke objek model melakukan penanganan even, serta melakukan proses render. Kelas-kelas ini masih dapat dikembangkan lagi oleh *Component Writer* untuk membuat komponen yang lebih khusus sesuai kebutuhan aplikasi.

Semua kelas Komponen Antarmuka JSF diturunkan dari kelas UIComponentBase, yang memberikan proses kerja dan *state* komponen *default.*

Tabel berikut adalah daftar kelas UIComponent yang disertakan pada rilis JSF ketika buku ini ditulis.

Tabel 2.1 Daftar Komponen User Interface

Kelas Komponen	Keterangan
UICommand	Merepresentasikan kontrol yang menghasilkan *action* ketika diaktifkan.
UIForm	Mengelompokkan sejumlah kontrol yang digunakan untuk mengirim data dari *client* ke server. Kelas ini analogi dengan tag form pada HTML.
UIGraphics	Menampilkan gambar.
UIInput	Merepresentasikan kontrol yang digunakan untuk menerima data dari pengguna.
UIOutput	Menampilkan data pada halaman web.
UIPanel	Menampilkan table.
UIParameter	Merepresentasikan parameter pada halaman web.
UISelectItem	Merepresentasikan sebuah nilai dari sejumlah nilai
UISelectItems	Merepresentasikan sejumlah nilai secara keseluruhan.
UISelectBoolean	Masukan bagi pengguna yang berupa data Boolean (true/false)
UISelectMany	Masukan bagi pengguna dimana pengguna dapat memilih subset dari sejumlah nilai
UISelectOne	Masukan bagi pengguna dimana pengguna dapat memilih satu dari sekian pilihan.

Biasanya *Page Author* dan *Application Developer* tidak secara langsung menggunakan kelas-kelas diatas, akan tetapi menggunakan tag yang bersesuaian dengan komponen tersebut. Sebagian besar komponen-komponen dapat direnderer

dengan cara yang berbeda, misalnya UICommand dapat di-*render* sebagai tombol atau sebagai *hyperlink.*

JSF RenderKit

Arsitektur komponen JSF didesain sedemikian rupa sehingga fungsionalitas komponen didefinisikan pada kelas-kelas komponen, sedangkan proses *render* komponen diserahkan ke *renderer* yang terpisah. Untuk menggunakan *renderer* disediakan pula tag yang bertugas menggambarkan hasil render komponen ke halaman web. Desain ini memiliki keunggulan sebagai berikut:

- Component Writers hanya perlu mendefinisikan karakteristik komponen sekali pada kelas-kelas komponen, dan membuat beberapa *renderer* yang berbeda yang bertanggung jawab bagaiman komponen tersebut ditampilkan pada halaman web. Sehingga representasi sebuah komponen tidak hanya berupa sebuah kontrol HTML.
- *Page Author* dan *Application developer* dapat mengubah tampilan dari komponen dengan memilih tag yang berbeda untuk memberikan representasi yang sesuai.

Sejumlah kelas komponen dipasangkan dengan *renderer* dan tag dalam sebuah *RenderKit*

Event dan Event Listener

Salah satu tujuan desain JavaServer Faces adalah memberikan paradigma pemrograman aplikasi berbasis web yang serupa dengan paradigma

pemrograman aplikasi desktop. Salah satu bentuk kesamaan tersebut adalah adanya model Event dan Event Listener di spesifikasi JavaServer Faces.

Even dihasilkan oleh komponen antarmuka ketika komponen tersebut diaktifkan. Even ini berisi informasi komponen apa yang membangkitkannya serta informasi tentang even yang terjadi. Untuk dapat memanfaatkan event ini, harus dibuat EventListener dan mendaftarkannya pada komponen yang akan menghasilkan event. Sebagai contoh, jika pada sebuah tombol diberikan EventListener, maka ketika tombol tersebut diklik ia akan membangkitkan sebuah even, dan JavaServer Faces akan menjalankan EventListener untuk memproses event tersebut.

```
<h:command_button id="custom" commandName="custom"
      commandClass="package-selected" label="change">
<f:action_listener type="cardemo.CarActionListener" />
</h:command_button>
```

Gambar 2.14 Contoh pemakaian event listener

Teknologi JavaServer Faces juga mendukung mekanisme validasi terhadap masukan dari pengguna. Untuk menggunakan validasi ini JSF menyediakan sejumlah tag yang berhubungan dengan validasi masukan

2.2.3.4. Navigation Model

Secara umum setiap aplikasi web terdiri dari sejumlah halaman web. Pada metode pengembangan konvensional, navigasi halaman didefinisikan secara langsung pada halaman web secara *hard-coded*. Untuk pengembangan aplikasi skala besar, cara ini akan menyulitkan pengembang ketika harus melakukan perubahan-perubahan navigasi halaman. Untuk penanganan navigasi JSF

memberikan solusi yang lebih fleksibel, yaitu dengan mendefinisikan navigasi halaman pada file konfigurasi *faces-config.xml*. Pada file ini didefiniskan kemana saja sebuah file dapat dihubungkan.Berikut adalah contoh sebuah cuplikan konfigurasi navigasi.

```
<navigation-rule>
  <from-tree-id>/ta_edit.jsp</from-tree-id>
  <navigation-case>
    <from-outcome>return</from-outcome>
    <to-tree-id>/ta_main.jsp</to-tree-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-tree-id>/ta_main.jsp</to-tree-id>
  </navigation-case>
</navigation-rule>
```

Gambar 2.15 Contoh konfigurasi navigasi pada faces-config.xml

Contoh di atas menggambarkan navigasi dari file /ta_edit.jsp. Untuk membuat sebuah hyperlink ke file ta_main.jsp dibuat sebuah komponen hyperlink dengan atribut action bernilai "return". Jika terjadi perubahan pada navigasi, misalnya akan diubah menjadi link ke file index.jsp, maka yang diubah hanya isi dari file konfigurasi dengan mengubah elemen to-tree-id menjadi index.jsp, sedangkan file ta_edit.jsp tidak berubah sedikitpun.

```
<h:command_hyperlink commandName="cmdReturn"
      action="return" label="kembali"/>
```

Gambar 2.16 Contoh pemakaian action untuk navigasi

2.2.3.5. Managed Bean Creation

Penggunaan *Managed Bean Creation* bertujuan untuk menghemat penggunaan resource ketika aplikasi membutuhkan JavaBean sebagai objek model. Berikut ini beberapa keunggulan *Managed Bean Creation* dibandingkan dengan penggunaan tag `<jsp:useBean>`.

- Pengembang dapat mendeklarasikan objek model pada file konfigurasi dan menggunakannya di seluruh aplikasi. Jika menggunakan `<jsp:useBean>` maka pada setiap file JSP yang menggunakan bean tersebut harus diberi deklarasi use:Bean agar JavaBean tersebut dapat diakses
- Jika terjadi perubahan pada JavaBean maka dapat diusahakan agar perubahan hanya dilakukan pada file konfigurasi, tidak perlu pada semua file JSP yang menggunakan JavaBean tersebut.
- Ketika sebuah managed bean dibuat, nilai awal properti dari JavaBean tersebut dapat di tentukan pada file konfigurasi.

```
<managed-bean>
      <managed-bean-name> MyManagedBean
            </managed-bean-name>
      <managed-bean-class>pkg01.MyBean
            </managed-bean-class>
      <managed-bean-scope> session </managed-bean-scope>
      <managed-property>
         <property-name>carImage</property-name>
         <value>current.gif</value>
      </managed-property>
</managed-bean>
```

Gambar 2.17 Contoh konfigurasi Managed Bean

Pada contoh di atas didefinisikan sebuah managed bean dengan nama `MyManagedBean`. Bean ini bertipe `pkg01.MyBean` dengan lingkup session. Properti yang diinisialisasi pada konfigurasi ini adalah properti `carImage` dengan nilai awal "current.gif". Untuk mengakses nilai dari Java Bean ini digunakan nama MyManagedBean. Contoh berikut adalah penggunaan dari managed bean untuk menampilkan gambar.

```
<h:graphic_image id="car_img" valueRef= "MyManagedBean.carImage"/>
```

Gambar 2.18 Contoh penggunaan Managed Bean

2.3. ORACLE TEXT

Pada bagian ini dijelaskan mengenai Oracle Text secara umum, berikut langkah-langkah pembuatan Aplikasi Query. Di bagian ini penulis banyak menggunakan istilah 'tabel teks' dan 'kolom teks'.Di sini 'tabel teks' artinya adalah tabel yang digunakan untuk menyimpan dokumen. Dimana pada tabel tersebut akan diberi indeks Oracle Text, yaitu pada kolom penyimpanan dokumen, kolom teks. Sedang yang dimaksud dokumen teks adalah dokumen yang akan diolah Oracle Text untuk diberi indeks. Versi Oracle Text yang digunakan penulis adalah Oracle Text Release 9.2.

2.3.1. Sekilas Oracle Text

Oracle Text adalah tool yang disediakan oleh Oracle untuk membantu developer membangun aplikasi *text query*, dan aplikasi *document classification.* Aplikasi *text query* adalah aplikasi yang mampu melakukan pencarian sebuah kata pada sejumlah dokumen teks. Dokumen teks dapat berupa sebuah file teks biasa, file dokumen berformat HTML, XML, atau Microsoft Word. Untuk dapat membuat aplikasi seperti ini, Oracle Text menyediakan dua jenis indeks database, yaitu CONTEXT dan CTXCAT. Setelah database memiliki Indeks ini, dapat dilakukan query terhadap indeks ini dengan operator CONTAINS atau CATSEARCH.

Aplikasi *document classification* adalah aplikasi yang mampu mengenali jenis dokumen-dokumen yang masuk ke database ke dalam sejumlah kelompok kategori yang telah ditentukan sebelumnya. Contoh aplikasi jenis ini misalnya

sebuah aplikasi Manajemen Arsip Berita yang mampu mengenali jenis berita yang dikandung oleh sebuah dokumen ketika dokumen itu akan dimasukkan ke database. Aplikasi dapat mengetahui apakah dokumen itu berisi berita politik, berita olahraga, atau berita kriminal. Jenis Indeks yang disediakan Oracle untuk menangani hal ini adalah indeks CTXRULE. Setelah Indeks ini dibuat, terhadap setiap dokumen yang masuk ke database, dapat dilakukan proses klasifikasi melalui query SQL dengan operator MATCHES. Indeks jenis ini hanya mendukung dokumen dengan format HTML, XML, serta file teks biasa.

Operator ABOUT

Pencarian yang dilakukan dengan Oracle Text tidak hanya berupa kata kunci secara tekstual, tetapi juga dapat berupa topik. Khusus untuk dokumen yang disimpan pada database berbahasa inggris atau perancis, operator ABOUT dapat digunakan untuk mencari dokumen dengan topik-topik tertentu. Sebagai contoh, jika dilakukan query tentang topik "olahraga", maka dokumen hasil pencarian mungkin saja tentang sepak bola, bulutangkis, dan lainnya. Dokumen hasil pencarian tidak harus mengandung kata "olahraga".

Kemampuan pencarian berdasarkan topik ini diperoleh dari Basis Pengetahuan yang dimiliki database. Basis Pengetahuan ini berupa sebuah daftar kata-kata yang disusun secara bertingkat berdasarkan klasifikasi temanya.

Document Services

Beberapa Fitur penting Oracle Text lainnya adalah fitur *document services* yang berupa FILTER, GIST, HIGHLIGHT, dan MARKUP.

- FILTER. Dengan fitur ini dapat diperoleh versi HTML atau versi teks dari sebuah dokumen terformat, misalnya dokumen berformat Microsoft Word.
- GIST. Fitur ini menghasilkan intisari dari dokumen berupa satu atau lebih paragraf yang paling merepresentasikan seluruh isi dokumen.
- HIGHLIGHT. Fitur ini menghasilkan nilai offset bagian-bagian dokumen yang memenuhi kriteria pencarian. Fitur ini nantinya akan menghasilkan sebuah tabel yang berisi posisi dari setiap kata yang dicari dalam dokumen.
- MARKUP. Fitur ini menghasilkan dokumen yang telah di-*markup* dalam format HTML atau teks biasa. Melalui tanda-tanda yang dihasilkan oleh fitur ini, pengguna dapat mengenali kata yang dicari. Dalam format HTML, dokumen akan lebih mudah ditelusuri dengan memberikan link navigasi pada kata yang dicari.

Secara umum untuk membuat sebuah aplikasi Query, dibutuhkan dua hal berikut:

1. Tabel teks yang sudah berisi dokumen
2. Sebuah indeks Oracle Text.

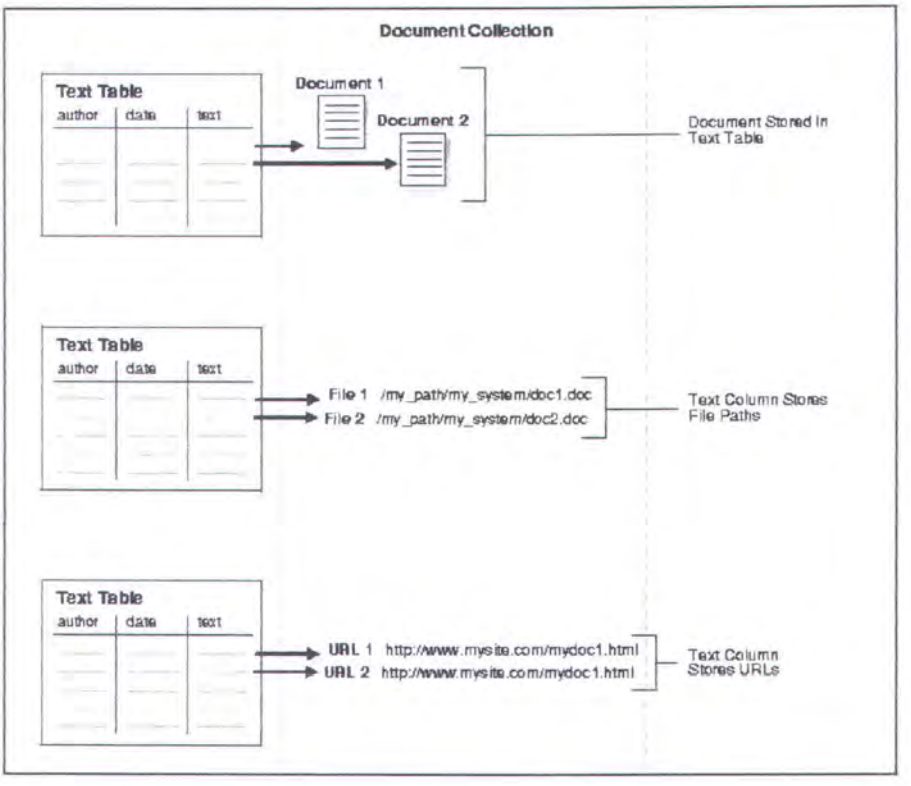
2.3.2. Membuat Tabel Teks.

Untuk aplikasi text query, ada 3 (tiga) cara dalam menyimpan dokumen pada database Oracle.

1. Dengan menyimpan dokumen langsung di database dalam tabel teks.
2. Dengan menyimpan informasi lokasi dokumen (*file path*) dalam tabel, sedangkan dokumennya tetap di *file system*.

3. Dengan menyimpan URL dari dokumen. Cara ini mengharuskan database terhubung ke jaringan internet, agar database dapat membaca isi dokumen langsung dari internet.

Oracle Text menentukan cara dokumen disimpan pada saat pembuatan indeks, yaitu melalui parameter DATASTORE dalam pernyataan SQL `create index`. Secara default, Oracle Text menganggap dokumen disimpan dengan cara pertama, yaitu langsung pada tabel.



Gambar 2.19 Tiga cara menyimpan dokumen untuk Oracle Text

Menyimpan Dokumen langsung di tabel

Cara ini dilakukan dengan tipe penyimpanan DIRECT_DATASTORE, dengan asumsi bahwa hanya satu kolom yang digunakan untuk menyimpan dokumen. Jika dokumen disimpan dalam beberapa kolom tabel, tipe

penyimpanannya adalah MULTI_COLUMN_DATASTORE. Dengan cara ini, ketika akan membuat indeks, Oracle akan menggabungkan kolom-kolom tersebut menjadi sebuah virtual document. Cara lainnya adalah dengan membuat relasi *master-detail* pada tabel. Dengan cara ini sebuah dokumen disimpan dalam sejumlah baris tabel. Tipe penyimpanan data yang digunakan untuk cara ini adalah DETAIL_DATASTORE. Selain itu, dokumen dapat pula disimpan dalam sebuah *nested table* dengan tipe penyimpanan NESTED_DATASTORE.

Menyimpan Dokumen melalui file path dan URL

Dengan dua cara ini, dokumen tidak secara langsung disimpan pada tabel, akan tetapi yang disimpan adalah informasi lokasi dokumen. Untuk penyimpanan melalui *file path*, digunakan tipe penyimpanan FILE_DATASTORE. Sedangkan untuk penyimpanan melalui URL, digunakan tipe penyimpanan URL_DATASTORE ketika membuat indeks.

2.3.3. Memuat dokumen ke dalam tabel

Berikut ini beberapa cara untuk memuat file dokumen ke dalam tabel di database Oracle: Melalui pernyataan SQL Insert, menggunakan SQL*Loader, menggunakan PL/SQL *Stored Procedure* `DBMS_LOB.LOADFROMFILE()`, serta melalui pemrograman Oracle Call Interface(OCI). Pada Bagian ini akan dijelaskan dua cara pertama.

- Melalui Pernyataan SQL insert

Cara pertama untuk memuat dokumen ke dalam tabel adalah dengan pernyataan SQL insert. Sebagai contoh, untuk memuat dokumen pada sebuah tabel `docs` dengan struktur berikut:

Tabel 2.2 Struktur tabel docs

Field	Tipe data
Id	NUMBER
Text	VARCHAR2

Digunakan pernyataan SQL Insert sebagai berikut:

```
INSERT into docs values(1, 'text of the first document');
INSERT into docs values(12, 'text of the second document');
```

Gambar 2.20 Pernyataan SQL Insert untuk memuat dokumen

- Menggunakan SQL*Loader

SQL*Loader adalah utilitas Oracle untuk memuat file ke dalam database. Sebagai masukan, SQL*Loader membutuhkan sebuah control file untuk menentukan spesifikasi proses pemuatan data ke dalam database serta satu atau lebih data file. Isi data file adalah data yang akan dimuat. Jika data yang ingin dimuat berupa sejumlah file, maka isi data file adalah path dari sejumlah file tersebut.

Dengan SQL*Loader, dokumen yang berada pada file sistem dapat dimuat ke dalam tabel. Nama Tabel yang dijadikan tujuan ditentukan pada control file yang menjadi argumen perintah SQL*Loader. Sedangkan nama-nama file yang akan dimuat ke dalam tabel ditentukan di dalam file lain.

Sebagai Contoh, sebuah tabel `search_table` yang akan diisi dengan sejumlah file dibuat dengan pernyataan SQL create tabel berikut:

```
create table search_table (
  tk          numeric primary key,
  title       varchar2(2000),
  text        clob
);
```

Gambar 2.21 Struktur tabel search_table

Perintah SQL*Loader yang digunakan untuk memuat file-file kedalam tabel adalah sebagai berikut:

```
% sqlldr userid=scott/tiger control=loader.ctl
```

Gambar 2.22 Perintah SQL*Loader untuk memuat file ke database

Perintah ini akan memuat dokumen pada *file system*, menggunakan *control file* loader.ctl, dengan akun pengguna "scott" dan kata sandi "tiger".

Isi *control file* pada perintah ini adalah sebagai berikut

```
LOAD DATA
        INFILE 'loader.dat'
        INTO TABLE search_table
        REPLACE
        FIELDS TERMINATED BY ';'
        (tk             INTEGER,
         title          CHAR,
         text_file      FILLER CHAR,
         text           LOBFILE(text_file)
TERMINATED BY EOF)
```

Gambar 2.23 Contoh control file

File control ini memberitahu SQL*Loader:

- Untuk memuat file-file yang namanya terdapat pada file 'loader.dat'.
- Sebagai tabel tujuan adalah tabel 'search_table'.
- Pemisah antar field pada file loader.dat adalah karakter ';'.

Sedangkan isi file loader.dat adalah sebagai berikut:

```
1;   Sun finds glitch in new UltraSparc III chip;0-1003-200-
5507959.html
2;   Redback announces loss, layoffs ;0-1004-200-5424681.html
3;   Cisco dumps acquired optical technology ;0-1004-200-
5510096.html
...
```

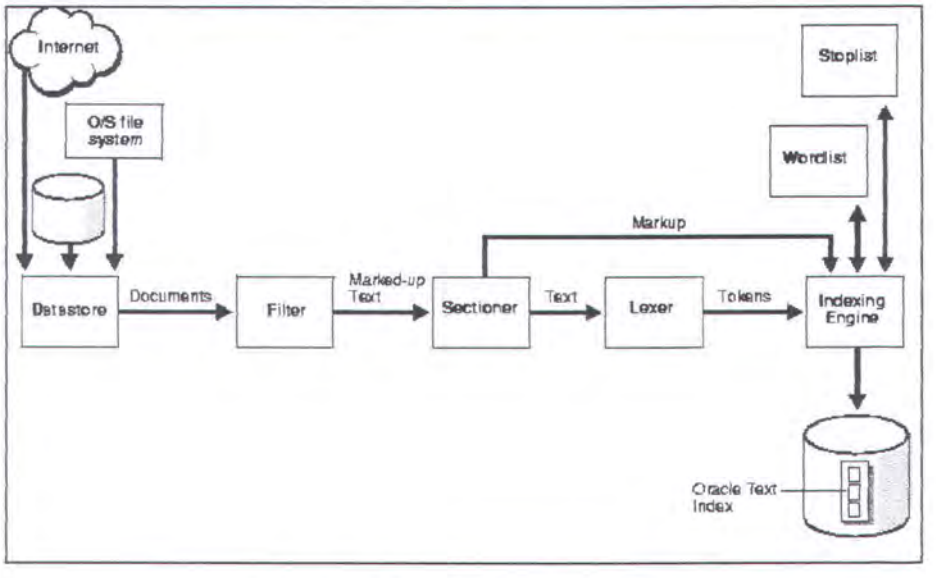
Gambar 2.24 Contoh isi data file

2.3.4. Membuat Index

Pada bagian ini dijelaskan beberapa hal mengenai pembuatan indeks Oracle Text, meliputi proses *indexing* Oracle Text, jenis-jenis indeks, serta cara pembuatan indeks CONTEXT

2.3.4.1. Proses Indexing

Yang dimaksud proses *indexing* di sini adalah proses penyusunan indeks token-token dari dokumen-dokumen yang tersimpan dalam tabel teks berdasarkan parameter dan nilai setting yang ditentukan. Proses ini diawali ketika eksekusi pernyataan SQL `CREATE INDEX`. Gambar berikut menunjukkan cara kerja proses *indexing* ini. Pada gambar dapat dilihat bahwa aliran data melalui beberapa proses pengolahan yang berbeda. Setiap objek pengolahan merepresentasikan nilai setting yang ditentukan pada pernyataan SQL `CREATE INDEX` atau `ALTER INDEX`.



Gambar 2.25 Proses Indexing Oracle Text

Objek Datastore

Pengolahan pertama aliran data dilakukan oleh objek Datastore. Objek ini membaca isi dokumen berdasarkan nilai setting Tipe penyimpanan data, yang berupa parameter DATASTORE pada pernyataan SQL CREATE INDEX. Sebagai contoh, jika nilai parameter DATASTORE adalah FILE_DATASTORE, maka database akan membaca isi dokumen di *file system* berdasarkan informasi lokasi dokumen di tabel teks.

Objek Filter

Proses selanjutnya dilakukan oleh objek FILTER, yang merepresentasikan parameter FILTER pada pernyataan SQL CREATE INDEX. Berikut beberapa bentuk pengolahan Filter berdasarkan nilai parameter:

- Tidak Ada proses pengolahan. Ini terjadi jika nilai parameter FILTER adalah NULL_FILTER. Filter ini sesuai untuk dokumen berformat HTML, XML dan teks biasa yang tidak membutuhkan pengolahan filter,
- Dokumen terformat difilter menjadi dokumen ter-*markup*. Nilai parameter untuk proses ini adalah INSO_FILTER. Dokumen berformat Microsoft Word, Microsoft PowerPoint dapat difilter dengan filter ini.
- Aliran data mengalami konversi set-karakter menjadi set-karakter yang dipakai oleh database. Nilai parameter untuk proses ini adalah CHARSET_FILTER.

Objek Sectioner

Proses berikutnya adalah proses pemilahan aliran data menjadi teks dan informasi *section* dokumen. Informasi *section* berupa lokasi awal dan akhir

section pada dokumen. Jenis *section* yang dibaca oleh *Sectioner* ditentukan pada nilai parameter SECTION GROUP.

Hasil dari proses ini yang berupa informasi *section* langsung diberikan ke *indexing engine*. Sedangkan data teks diteruskan ke Objek Lexer.

Objek Lexer

Objek Lexer memecah data teks menjadi token-token berdasarkan *setting* bahasa yang digunakan database. Dalam mengekstrak token, Oracle Text menggunakan nilai *setting* parameter LEXER. Parameter ini menentukan aturan yang digunakan Lexer dalam memecah teks menjadi token, misalnya karakter pemisah yang digunakan, atau bagaimana token-token itu disimpan, apakah diubah kedalam huruf besar semua tetap sebagai mana aslinya.

Jika *theme indexing* diaktifkan, maka Lexer akan menganalisa teks untuk mencari token-token berdasarkan topik. Token-token ini nantinya digunakan dalam membuat indeks topik.

Indexing Engine

Indexing Engine membuat *inverted index* yang menghasilkan *mapping* dari token ke dokumen. Pada proses ini, Oracle Text membaca *stoplist* untuk menjaga agar tidak ada kata-kata di dalam *stopwords* dan *stopthemes* yang dimasukkan ke indeks. Jika setting WORDLIST diaktifkan, Oracle Text juga akan membaca nilai setting ini untuk menentukan aturan pembuatan indeks *prefix*dan *suffix*.

2.3.4.2. Jenis-Jenis Indeks Oracle Text

Di awal pembahasan Oracle Text telah dikenalkan tiga jenis indeks yang dapat dibangun untuk membuat aplikasi *text query* atau aplikasi *document classification*. Di sini akan dijelaskan lebih jauh tentang ketiga jenis indeks tersebut, perbedaan-berbedaan antara ketiganya, serta pertimbangan dalam memilih indeks yang sesuai.

Secara umum pemilihan indeks Oracle Text adalah sebagai berikut:

- Indeks CONTEXT digunakan untuk membangun aplikasi *text query* yang menyimpan dokumen-dokumen yang berukuran besar dan koheren. Indeks ini dapat mengenali berbagai format dokumen seperti Microsoft Word, HTML, XML ataupun teks biasa.
- Indeks CTXCAT digunakan untuk meningkatkan kinerja pencarian untuk query yang tidak hanya mencari kata pada isi dokumen, tetapi juga pada field-field lain tabel teks. Dengan kata lain indeks ini dikhususkan untuk pencarian dengan *mixed query*. Misalnya memadukan pencarian dalam dokumen dengan kriteria seperti judul dokumen, penulis dokumen yang disimpan pada field yang berbeda. Bentuk dokumen yang disimpan berupa dokumen berukuran relatif kecil atau potongan kecil dokumen.
- Indeks CTXRULE digunakan untuk membuat aplikasi *document classification*. Indeks ini dapat dibuat pada tabel yang memuat sejumlah query, dimana setiap query memiliki klasifikasi tersendiri. Sebuah dokumen dapat diklasifikasikan dengan operator MATCHES.

Pada Tabel 2.3 di halaman berikut diberikan rincian tipe-tipe indeks Oracle Text.

Tabel 2.3 Perbandingan Jenis Indeks Oracle Text

Tipe Indeks : CONTEXT	
Penjelasan	digunakan untuk membangun aplikasi *text query* yang menyimpan dokumen-dokumen yang berukuran besar dan koheren. Indeks ini membutuhkan proses sinkronisasi CTX_DDL.SYNC_INDEX sesudah setiap query DML (Insert, update atau delete). Indeks ini mendukung semua fitur *document services* dan *query services* pada Oracle Text. Indeks ini mendukung tabel teks berpartisi.
Parameter dan Setting yang didukung	Semua jenis *preference* dan parameter, kecuali INDEX SET
Operator Query	CONTAINS. Indeks ini memiliki grammar CONTEXT yang memiliki cukup banyak operator.
Tipe Indeks : CTXCAT	
Penjelasan	Indeks ini dikhususkan untuk pencarian dengan *mixed query*. Misalnya memadukan pencarian dalam dokumen dengan kriteria seperti judul dokumen, penulis dokumen yang disimpan pada field yang berbeda. Bentuk dokumen yang disimpan berupa dokumen berukuran relatif kecil atau potongan kecil dokumen. Indeks ini bersifat transaksional, tidak membutuhkan sinkronisasi dengan CTX_DDL.SYNC_INDEX sesudah setiap query DML. Karena secara otomatis akan melakukan *update* setiap setelah query DML. Tidak mendukung partisi tabel dan indeks. Tidak mendukung fitur *document services* (highlighting, markup, themes, gists)

	Tidak mendukung fitur *query services* (explain, query feedback, browse words.)
Parameter dan Setting yang didukung	INDEX SET, LEXER (tanpa *theme indexing*), STOPLIST STORAGE, WORDLIST (untuk data dalam bahasa Jepang, hanya mendukung atribut prefix_index), Tidak mendukung *Format, charset*, dan *language columns.* Tidak mendukung partisi tabel dan partisi indeks.
Operator Query	CATSEARCH. Memiliki grammar CTXCAT, yang mendukung operasi logika, query frase, serta *wildcard.*
Tipe Indeks : CTXRULE	
Penjelasan	Digunakan untuk membuat aplikasi *document classification.* Indeks ini dapat dibuat pada tabel yang memuat sejumlah query, dimana setiap query mendefinisikan kriteria klasifikasi. Dokumen diklasifikasikan menggunakan operator MATCHES, hingga menemukan baris query yang kriterianya dipenuhi oleh dokumen tersebut.
Parameter dan Setting yang didukung	Tipe LEXER yang didukung hanya BASIC_LEXER, sehingga dalam memilah teks menjadi token, bahasa yang dikenali hanya bahasa Inggris serta sebagian besar bahasa Eropa Barat, serta menggunakan *white space* sebagai karakter pemisah. Tidak mendukung parameter FILTER, MEMORY, DATASTORE, dan POPULATE.
Operator Query	MATCHES. Mendukung operator ABOUT, STEM, AND, NEAR, NOT, dan OR. Tidak mendukung operator ACCUM, EQUIV, WITHIN, WILDCARD, FUZZY, SOUNDEX, MINUS, WEIGHT, THRESHOLD.

Tipe Indeks : CTXXPATH	
Penjelasan	digunakan untuk mengoptimalkan query ExistsNode() pada kolom bertipe XMLType.
Parameter dan Setting yang didukung	STORAGE
Operator Query	Menggunakan query ExistsNode()

2.3.4.3. Pembuatan Indeks CONTEXT

Secara default indeks yang dibuat Oracle Text memiliki Tipe Penyimpanan (DATASTORE) langsung pada tabel database, atau DIRECT_DATASTORE. Jika setting ini dipenuhi, Indeks dapat langsung dibuat tanpa mengisikan nilai parameter pernyataan SQL CREATE INDEX. Secara otomatis, sistem akan mendeteksi bahasa yang digunakan, tipe data kolom teks, format dokumen, dan menyesuaikan sejumlah *preference* lainnya.

Untuk membuat sebuah Indeks Oracle Text, langkah-langkahnya adalah: menentukan *preference* indeks yang akan dibuat, membuat *preference* yang telah ditentukan, serta membuat indeks melalui pernyataan SQL CREATE INDEX dengan argument *preference* yang telah dibuat.

Menentukan *preference* indeks

Sebelum membuat indeks, terlebih dahulu harus ditentukan nilai setting *preference* dari indeks yang akan dibuat. Tabel pada halaman berikut memberikan penjelasan *setting* yang harus ditetapkan ketika akan membuat indeks.

Tabel 2.4 Penjelasan Setting Indeks Oracle Text

Setting	**Penjelasan**
Datastore	Bagaimana dokumen disimpan?
Filter	Bagaiman dokumen dikonversi menjadi teks biasa?
Lexer	Dengan bahasa apa indeks akan dibuat?
Wordlist	Bagaimana operator stem dan fuzzy menemukan referensi pencarian kata?
Storage	Bagaimana data indeks disimpan?
Stop List	Kata-kata dan topik apa saja yang tidak perlu diindeks?
Section Group	Bagaimana *section* dokumen didefinisikan?

Membuat preference

Pada bagian berikut akan diberikan beberapa contoh cara membuat *preference* dari indeks. *Preference* yang dibuat adalah DATASTORE, FILTER, LEXER, dan SECTION GROUP.

- DATASTORE

Membuat DIRECT_DATASTORE

Contoh berikut membuat sebuah tabel teks, lalu mengisinya sebanyak dua baris, serta membuat indeks dengan tipe DIRECT_DATASTORE. Karena tipe ini merupakan tipe default, maka tipe ini dapat juga dibuat dengan nilai DATASTORE DEFAULT_DATASTORE.

```
create table mytable(id number primary key, docs clob);
insert into mytable values(111555,'this text will be indexed');
insert into mytable values(111556,'this is a direct_datastore
example');
commit;
create index myindex on mytable(docs) indextype is ctxsys.context
  parameters ('DATASTORE CTXSYS.DEFAULT_DATASTORE');
```

Gambar 2.26 Membuat Indeks dengan tipe DIRECT_DATASTORE

Membuat URL_DATASTORE dan FILE_DATASTORE

Pada contoh berikut dibuat *preference* my_url yang menggunakan tipe URL_DATASTORE, dan *preference* my_file yang menggunakan tipe FILE_DATASTORE. Preference ini nantinya dapat digunakan sebagai argumen pernyataan SQL CREATE INDEX sebagaimana pada gambar 2.25 di atas.

```
begin
 ctx_ddl.create_preference('my_url','URL_DATASTORE');
 ctx_ddl.set_attribute('my_url','HTTP_PROXY','www-
proxy.us.oracle.com');
 ctx_ddl.set_attribute('my_url','NO_PROXY','us.oracle.com');
 ctx_ddl.set_attribute('my_url','Timeout','300');


 ctx_ddl.create_preference('my_file', 'FILE_DATASTORE');
 ctx_ddl.set_attribute('my_file', 'PATH', '/docs');
end;
```

Gambar 2.27 Membuat preference dengan tipe URL_DATASORE dan FILE_DATASTORE.

- FILTER

Pada contoh berikut dibuat *preference* pada tabel docs yang menggunakan NULL_FILTER serta *section group* yang mengenali *section* berupa tag HTML, yaitu ctxsys.html_section_group. Preference yang menggunakan INSO_FILTER dapat dibuat dengan cara ini.

```
create index myindex on docs(htmlfile) indextype is
ctxsys.context
  parameters('filter ctxsys.null_filter');
```

Gambar 2.28 Membuat preference dengan NULL_FILTER

- LEXER

Salah satu setting LEXER yang dapat disesuaikan nilainya adalah setting karakter Printjoins. Karakter Printjoins adalah karakter di luar huruf dan angka yang akan tetap dimasukkan sebagai bagian dari indeks. Dengan cara ini kata

"web-site" tetap dimasukkan kedalam indeks sebagai "web-site", tidak dipisah menjadi 'web' dan 'site'.

Contoh berikut akan membuat *preference* bernama "mylex" dengan mengisikan nilai setting karakter Printjoins berupa karakter garisbawah dan tanda hubung "_-" sebagai sebuah Printjoins dengan tipe BASIC_LEXER. Setelah itu dibuat indeks menggunakan *preference* tersebut.

```
begin
ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
end;

create index myindex on mytable ( docs )
  indextype is ctxsys.context
  parameters ( 'LEXER mylex' );
```

Gambar 2.29 Membuat preference Printjoins dengan BASIC_LEXER

- SECTION GROUP

Pada dokumen XML atau HTML, Oracle Text menyediakan fitur untuk mencari kata tertentu pada *section* tertentu yang didefinisikan sebelumnya. Sebagai contoh, untuk dokumen HTML, dapat dilakukan pencarian khusus pada bagian *title* dokumen, setelah sebelumnya mendefinisikan *section* yang mengenali tag `<title>` pada dokumen HTML. Contoh Gambar 2.29 berikut mendefinisikan sebuah *section* dokumen HTML berupa tag `<H1>`.

```
begin
ctx_ddl.create_section_group('htmgroup',
'HTML_SECTION_GROUP');
ctx_ddl.add_zone_section('htmgroup', 'heading', 'H1');
end;
```

Gambar 2.30 Membuat section group untuk tag HTML.

Membuat Indeks dengan SQL CREATE INDEX

Setelah menentukan *preference* serta membuat *preference* melalui pernyataan PL/SQL, langkah selanjutnya adalah membuat Indeks dengan pernyataan SQL CREATE INDEX.

- Membuat Indeks CONTEXT default

Contoh berikut membuat sebuah indeks pada kolom "text" di tabel "docs" dengan tipe CONTEXT tanpa mengisi parameter pernyataan CREATE INDEX.

```
CREATE INDEX myindex ON docs(text) INDEXTYPE IS CTXSYS.CONTEXT;
```

Gambar 2.31 Membuat Indeks CONTEXT default

Pada setting default, Oracle Text akan membuat indeks dengan:

- ✓ Diasumsikan dokumen disimpan secara langsung pada table text. Tipe data kolom teks dapat berupa CLOB, BLOB, BFILE, VARCHAR2, atau CHAR.
- ✓ Mendeteksi tipe data kolom teks, serta menggunakan filter jika tipe data kolom adalah biner. Jika tipe data kolom adalah teks, maka tidak digunakan filter.
- ✓ Menggunakan setting bahasa sesuai setting bahasa database
- ✓ Menggunakan stoplist default yang disediakan database.
- ✓ Mengaktifkan fitur fuzzy dan stemming, jika setting bahasa yang dipakai mendukung fitur ini.

- Membuat Indeks CONTEXT dengan *preference*

Contoh gambar 2.32. berikut membuat indeks CONTEXT menggunakan *preference* yang telah dibuat sebelumnya.

```
create index myindex on docs(htmlfile) indextype is
ctxsys.context
parameters('datastore my_url filter ctxsys.inso_filter
section group htmgroup lexer mylex');
```

Gambar 2.32 Membuat indeks CONTEXT dengan preference

Contoh di atas membuat indeks CONTEXT dengan *preference* tipe penyimpanan data pada *file system* (preference my_url, gambar 2.27), menggunakan filter INSO_FILTER, menggunakan *section group* htmgroup (gambar 2.30), dilengkapi dengan *preference* LEXER mylex(gambar 2.29.)

2.3.5. Menyusun Query

Setelah Indeks yang dibutuhkan dibuat sesuai *preference* kebutuhan aplikasi, maka terhadap tabel terindeks telah dapat dilakukan proses query.

Pada bagian ini akan dijelaskan bagaimana menyusun query Oracle Text. Namun hanya dibatasi pada operator CONTAINS karena tipe indeks yang digunakan dalam tugas akhir ini, adalah indeks CONTEXT. Penjelasan tentang indeks dan query untuk tipe indeks lainnya dapat dilihat pada tabel 2.3.

Melakukan Query dengan operator CONTAINS

Terhadap sebuah indeks CONTEXT dapat dilakukan query dengan operator CONTAINS. Dalam mendefinisikan kriteria pencarian operator CONTAINS juga memiliki sejumlah operator dasar. Dengan operator-operator tersebut dapat dilakukan pencarian menggunakan kriteria logika, *fuzzy*, *stemming*, *thesaurus*, serta pencarian dengan *wildcard*. Selain itu untuk dokumen HTML dan XML dapat dilakukan pencarian pada struktur dokumen tertentu. Untuk pencarian berdasarkan topik yang dikandung dokumen, dapat digunakan operator ABOUT.

Menggunakan operator SCORE

Penggunaan operator CONTAINS berhubungan dengan penggunaan operator SCORE. Operator CONTAINS memiliki nilai kembalian berupa skor yang dimiliki dokumen dalam hal jumlah kata yang ditemukan. Nilai tertinggi nilai skor ini adalah 100, sedangkan nilai terendah adalah 0. Semakin banyak jumlah kata yang ditemukan maka skor dari Operator CONTAINS akan semakin tinggi. Nilai skor ini berbanding terbalik dengan jumlah set dokumen pencarian. semakin kecil jumlah set dokumen pencarian, semakin tinggi jumlah kata yang harus ditemui pada dokumen untuk mencapai skor 100. Gambar 3.33 berikut memberikan contoh penggunaan operator CONTAINS dan SCORE.

```
SELECT SCORE(1), title from news
          WHERE CONTAINS(text, 'oracle', 1) > 0
          ORDER BY SCORE(1) DESC;
```

Gambar 2.33 Penggunaan operator CONTAINS dan SCORE.

Pada query gambar 2.33 dilakukan pencarian terhadap kata 'oracle' di kolom `text` pada tabel `news`. Kriteria CONTAINS(...) > 1 menandakan bahwa hasil pencarian haruslah dokumen yang memiliki skor di atas 0 atau dengan kata lain meniadakan dokumen yang tidak mengandung kata kunci pencarian. Argumen ketiga operator CONTAINS merupakan angka untuk mengidentifikasi operator CONTAINS. Angka ini digunakan pada operator SCORE( ) untuk menentukan operator CONTAINS yang mana yang akan diambil nilai skornya.

Menggunakan operator ABOUT

Operator about digunakan untuk pencarian topik. Oracle Text akan mencari dokumen yang mengandung kata yang berhubungan dengan kata kunci topik yang dicari. Query berikut memberi contoh penggunaan operator about.

```
SELECT SCORE(1), title FROM news
          WHERE CONTAINS(text, 'about(education)', 1) > 0
          ORDER BY SCORE(1) DESC;
```

Gambar 2.34 Penggunaan operator about

Dengan query di atas akan dihasilkan sejumlah dokumen yang mengandung topik edukasi. Dokumen tersebut tidak harus mengandung kata "education", tetapi mungkin mengandung kata "school", "university", "lecturer", dan lain sebagainya.

2.4. ARSITEKTUR MODEL 2X

Sub bab ini berisi ulasan tentang konsep arsitektur Model 2X pada aplikasi berbasis web. Menjelaskan arsitektur sistem termasuk komponen-komponen yang menyusunnya. Pada bagian awal akan dijelaskan perkembangan arsitektur aplikasi web, dilanjutkan Arsitektur MVC sebagai dasar dari Model 2X dan pada bagian ketiga akan dijelaskan tentang Model 2X.

2.4.1. Definisi Arsitektur

Dalam pembahasan Arsitektur Aplikasi ini, penulis menggunakan definisi arsitektur sebagai berikut:

Sebuah arsitektur merupakan sejumlah keputusan signifikan mengenai pengorganisasian sistem perangkat lunak, pemilihan elemen struktural dan antarmukanya yang akan digunakan untuk membuat sistem, beserta perilakunya

seperti yang sudah ditentukan dalam kolaborasi antar elemen, penggabungan dari elemen struktural dan perilaku menjadi subsistem yang lebih besar secara progresif, dan tipe arsitektur yang menunjukkan organisasi ini---elemen-elemen beserta antarmukanya, kolaborasinya, dan komposisinya [BOO99].

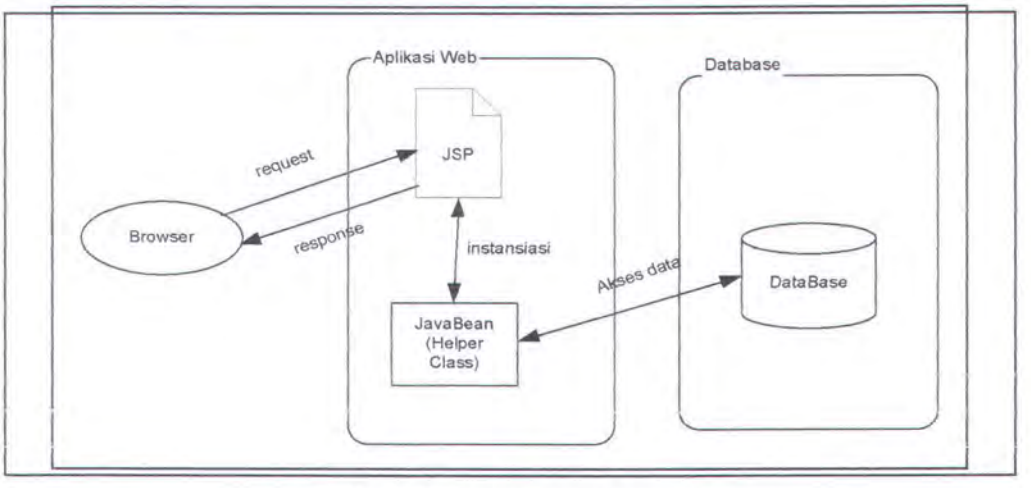
2.4.2. Perkembangan Arsitektur Aplikasi Web

Ketika pertama kali dirilis, JSP memberikan perubahan besar bagi para pengembang aplikasi web. hal ini dikarenakan kemudahan yang diberikan oleh JSP kepada pengembang dalam membangun aplikasi web. Sebelum ditemukannya teknologi JSP, untuk membangun aplikasi berbasis web para pengembang Java harus menggunakan servlet. Servlet dalam pemakaiannya lebih mirip dengan sebuah program Java daripada sebuah halaman web atau HTML. Karena memang pada dasarnya sebuah servlet adalah sebuah kelas Java yang diturunkan dari kelas `javax.servlet.HttpServlet` (khusus untuk protokol HTTP) atau `javax.servlet.GenericServlet` (Untuk semua protokol). Kelas ini memiliki method `doGet()` atau `doPost()` atau `service()`. Disinilah kode program java harus dibuat agar dapat menghasilkan tampilan yang diinginkan. Sehingga untuk operasi-operasi yang berinteraksi dengan kelas-kelas lain sangat mudah dilakukan dari sebuah servlet. Sedangkan jika yang akan dihasilkan adalah sebuah halaman web, maka pengambang aplikasi harus menuliskan semua kode HTML ke dalam servlet. Tentunya hal ini akan menyulitkan pengembang jika halaman web yang dibuat tidak sederhana.

Dengan JSP pengembang dapat langsung menuliskan kode HTML di dalam file jsp dan menyelipkan kode program dalam bentuk scriplet. yang diapit

oleh tag <% dan %>. Kemudahan ini memberikan perubahan yang signifikan dalam hal metode pengembangan aplikasi web.

Pada fase ini pola pengembangan yang banyak dipakai adalah metode *page-centric*. Di mana setiap *request* dari *client* akan diarahkan langsung ke halaman JSP yang sekaligus akan mengolah data yang masuk dari pengguna dan menyusun tampilan halaman web. Skema dari pola ini dapat dilihat pada Gambar 2.35. Pada pola pengembangan seperti ini kode program akan tersebar di sejumlah besar file JSP, baik kode program yang berkaitan dengan proses bisnis ataupun kode program yang sekedar mengatur tempilan halaman web. Keuntungan pola pengembangan ini adalah kecepatan pengembangan dan kesederhanaan aplikasi. Kode program dapat diletakkan dimana saja kode itu dibutuhkan.



Gambar 2.35 Skema Desain *Page-centric*

Untuk sebuah aplikasi berbasis web dengan skala kecil, metode pengembangan seperti ini memudahkan pengembang dalam bekerja. Akan tetapi untuk aplikasi dengan skala menengah-besar cara ini lambat laun akan menyulitkan pengembang karena akan menimbulkan permasalahan *sphagetti*

code, dimana kode program java bercampur dengan kode HTML sehigga menjadi sulit terbaca.

2.4.3. Arsitektur Model-View-Controller

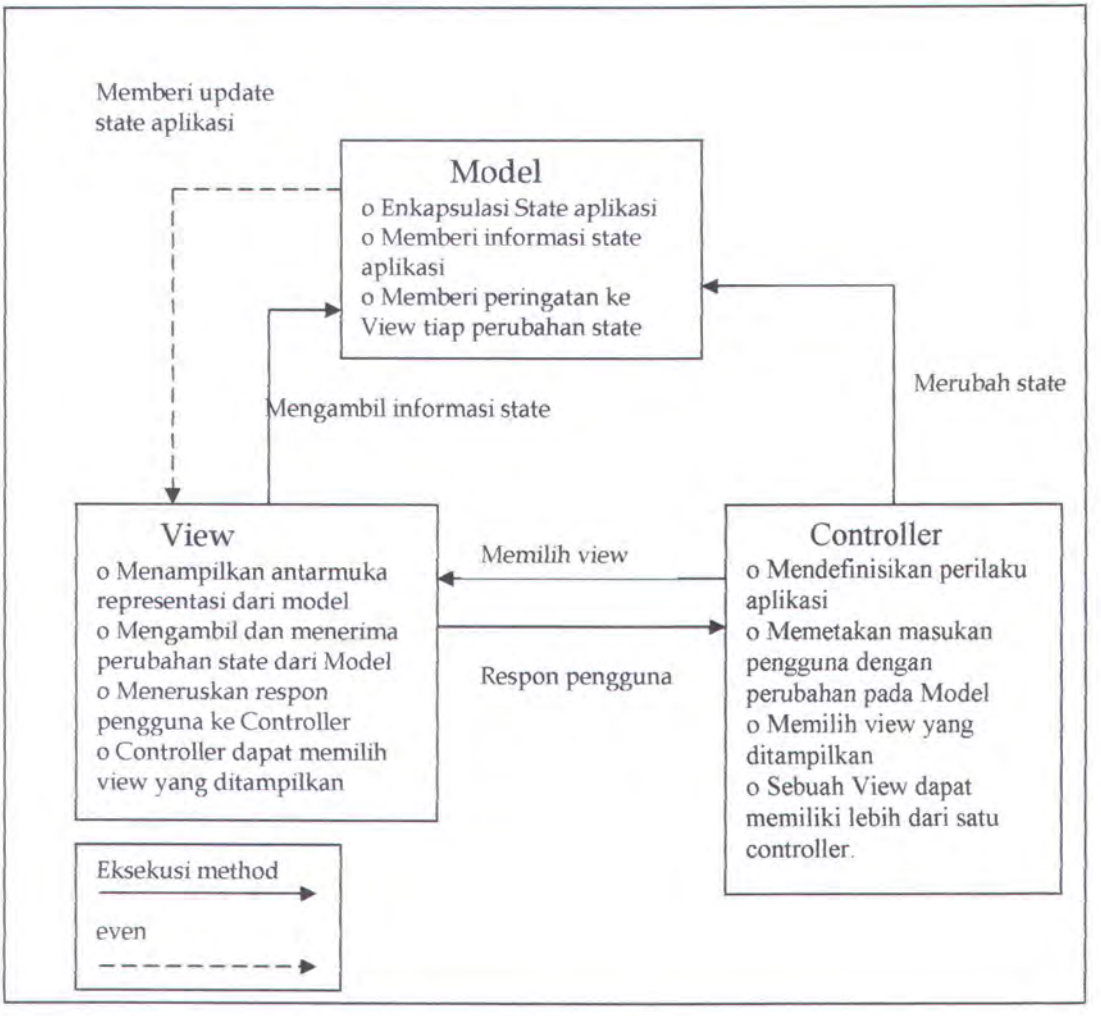
Arsitektur MVC ditemukan pada bahasa pemrograman SmallTalk. Arsitektur ini memecah sebuah aplikasi menjadi tiga tipe objek, yaitu *model, view,* dan *controller*. Arsitektur MVC dapat dikatakan sebagai sebuah desain tingkat tinggi dari aplikasi, sebab arsitektur ini mengandung sejumlah pola perancangan tingkat rendah.

Beberapa implementasi umum dari arsitektur ini adalah pemisahan antara antarmuka dengan implementasi proses bisnis pada Form Windows dalam bentuk pemrograman *event-driven*. Implementasi proses bisnis diletakkan pada *event handler,* dalam bentuk kode program. Sedangkan perancangan antarmuka dilakukan di Form Designer Microsoft Visual Studio. Sedangkan pada pemrograman java, dikenal adanya objek Event, Action, ActionListener sebagai bentuk pemrograman *event-driven.*

Berikut ini penjelasan dari elemen-elemen MVC:

- Model, merepresentasikan data aplikasi dan proses bisnis yang memanipulasi data.
- View, memberikan representasi visual dari model. Dapat berupa Form windows, halaman web, atau form antarmuka PDA. Kelebihan utama dari MVC adalah kemampuan untuk memberikan beberapa view sekaligus untuk menampilkan data yang sama, dimana masing-masing view memiliki format dan teknologi yang berbeda.

- Setiap view memiliki sebuah controller. Controller memisahkan antara representasi visual dan implementasi proses bisnis. Controller juga menangani langsung interaksi dengan pengguna serta mengendalikan akses ke model. Tergantung masukan dari pengguna, controller dapat merubah view yang ditampilkan, mengubah state aplikasi pada model, atau menampilkan view yang berbeda.



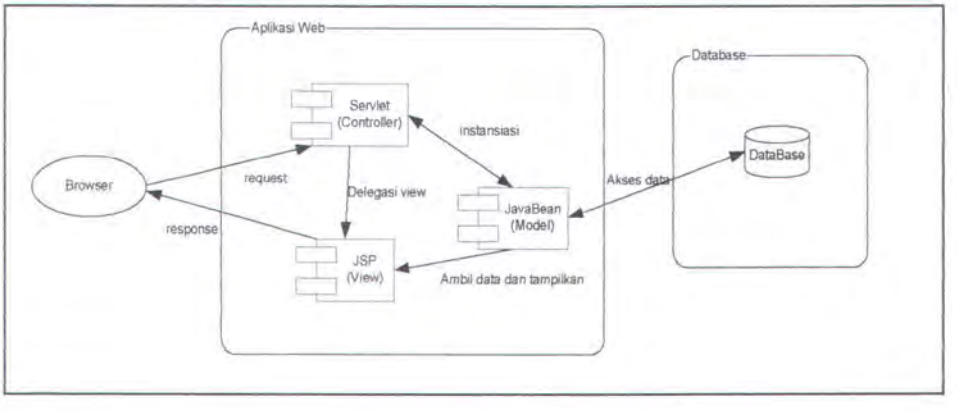
Gambar 2.36 Skema Arsitektur Model-View-Controller.

2.4.3.1. Keuntungan Arsitektur MVC

Arsitektur MVC dibangun untuk aplikasi yang memiliki beberapa komponen view. MVC memberikan solusi permasalahan kode program yang sulit ditata, (biasanya terjadi pada aplikasi skala besar) dengan memecahnya menjadi beberapa komponen(model, view, dan controller). Dengan menggunakan pemrograman berorientasi objek serta perancangan pola yang tepat, aplikasi menjadi lebih mudah baik dalam perancangan, implementasi, maupun perawatan.

2.4.3.2. Implementasi Model-View-Controller pada aplikasi web

Untuk implementasi arsitektur MVC pada aplikasi berbasis web, terdapat perbedaan dibandingkan dengan aplikasi *desktop*. Penerapan arsitektur **MVC** (Model, View, Controller) dilakukan dengan pembagian peran sebagai berikut: JavaBeans berfungsi sebagai model, JSP sebagai View atau layer presentasi, dan Servlet sebagai Controller. Tidak setiap view memiliki controller, tetapi hanya ada sebuah komponen controller yang berupa servlet utama yang akan mengkoordinasikan antara View dan Model. Desain seperti ini populer dengan nama **JSP Model2** (Lihat Gambar 2.37), dengan asumsi bahwa model pengembangan sebelumnya (*page-centric*) disebut **JSP Model1**.



Gambar 2.37 Desain Model-View-Controller pada aplikasi web berbasis java

Dengan arsitektur ini, request yang datang diarahkan ke servlet yang berfungsi sebagai controller. Request ini kemudian diteruskan ke halaman JSP atau JavaBeans yang sesuai. Bagian Model dari Arsitektur MVC mewakili internal state dari sistem (berisi data) serta Actions (berisi proses bisnis). Actions dieksekusi untuk mengubah state dari sistem. Dalam konteks bahasa, internal state analogi dengan kata benda, sedangkan Actions analogi dengan kata kerja.

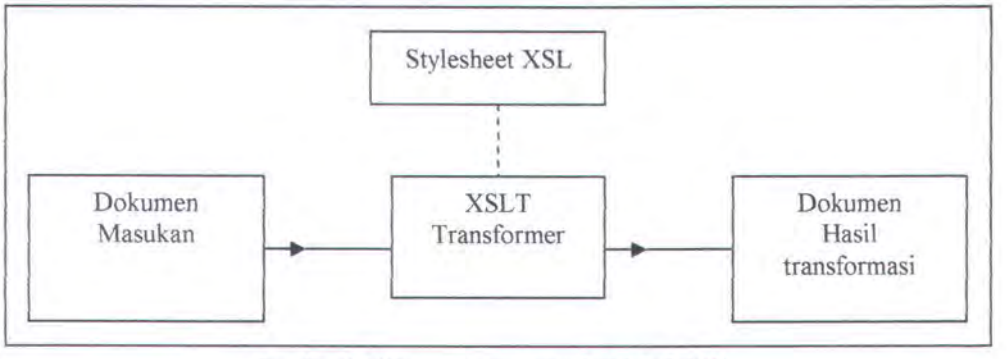
Untuk merepresentasikan **Model** digunakan JavaBeans dimana properti JavaBean tersebut mewakili detil dari state sistem. Untuk merepresentasikan Actions juga dapat digunakan kelas java yang khusus berisi proses bisnis yang berkaitan dengan Actions. Jika sistem yang dibangun memiliki skala kecil, Actions dapat diujudkan dalam bentuk sebuah method yang terdapat pada JavaBeans internal state. Bagian **View** dapat dibangun dengan JSP yang nantinya menghasilkan kode-kode HTML. Bagian **Controller** bertanggung jawab mengatur jalannya aplikasi. Controller menerima request dari client dan menentukan proses bisnis mana yang bertanggung jawab untuk mengolah request yang datang. Selanjutnya Controller mendelegasikan request kepada bagian View untuk menampilkan hasil pengolahan request. Controller dapat diimplementasikan dalam bentuk Servlet ataupun JSP, namun penggunaan Servlet sebagai controller lebih umum.

2.4.4. Konsep Arsitektur Model 2X

Pada beberapa kasus format luaran yang berupa HTML tidak sepenuhnya dibutuhkan. Sebagai contoh, ada kalanya dibutuhkan pemisahan antara proses penambahan layout tampilan dengan proses lokalisasi. Demikian pula tidak

menutup kemungkinan dibutuhkan dukungan aplikasi terhadap beberapa jenis *devices* dengan sumber data aplikasi yang sama. Pada kasus-kasus seperti ini dibutuhkan proses transformasi luaran aplikasi sebelum luaran tersebut dikirim ke *client*. Disini penggunaan XML/XSLT sesuai untuk melakukan transformasi luaran sebelum hasil bagian View dari arsitektur MVC dikrim ke *client*. Paduan antara Arsitektur MVC dan proses transformasi XML/XSLT inilah yang kemudian diwujudkan dalam bentuk Arsitektur Model 2X.

Arsitektur Model 2X sebagai pengembangan lebih lanjut dari Model2/MVC memiliki tambahan layer presentasi yang lebih modular dan fleksibel dibandingkan Model 2. Pada Model 2 luaran dari bagian View berbentuk halaman web HTML dan langsung dikirim ke *client*. Sedangkan pada Model 2X luaran yang dihasilkan oleh Servlet atau JSP berbentuk data XML. Selanjutnya data XML ini akan melalui proses transformasi sebelum dikirim ke browser (lihat Gambar 2.37). Proses transformasi ini dilakukan oleh *XML Processor*. Salah satu contoh XML *processor* adalah XSLT. Dengan XSLT dimungkinkan adanya kostumisasi tampilan terhadap data XML sehingga menghasilkan beberapa *style* untuk *content* XML yang sama.

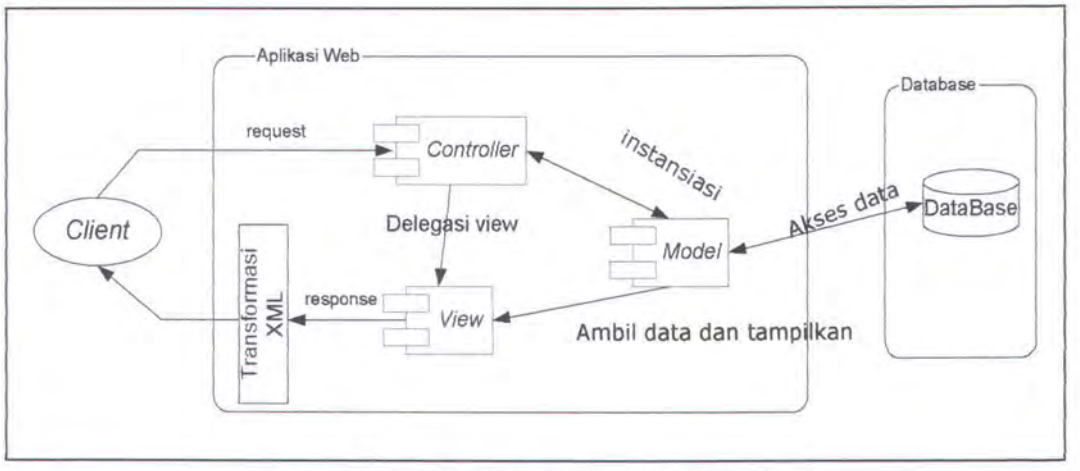


Gambar 2.38 Layer presentasi dari Model 2X

2.4.4.1. Gambaran Umum Arsitektur Model 2X

Pada bagian ini diberikan gambaran umum dari Arsitektur Model 2X.

Arsitektur ini terdiri dari dua komponen pokok, yaitu MVC (Model, View dan Controller) dan Transformasi XML.



Gambar 2.39 gambaran umum Arsitektur Model 2X

Sebagaimana yang telah dijelaskan pada bab 2.4.2, Arsitektur MVC terdiri dari 3 (tiga) buah komponen, yaitu:

- Model, berisi data dan proses bisnis. Bagian ini merepresentasikan kondisi dari sistem. Untuk mengakses database Model dapat berinteraksi langsung dengan database ataupun melalui data layer.
- View sebagai presentation layer. View bertugas menampilkan informasi yang diperoleh dari Model.
- Controller, bertugas mengatur jalannya aplikasi. Pada aplikasi web Controller menerima request dari client, memanggil *Model* yang akan menangani request, dan meneruskan request ke *View*.

Dalam Arsitektur Model 2X, luaran dari bagian MVC ditransformasi terlebih dahulu melalui proses transformasi XML sebelum dikirim ke *client*. Dengan demikian keluaran dari bagian MVC harus memenuhi format XML.

Komponen transformasi XML mengolah hasil keluaran ini menjadi format lain sesuai kebutuhan sistem. Proses transformasi dapat berlangsung satu kali atau beberapa kali. Jika terjadi beberapa kali, proses ini disebut sebagai *XML Pipeline*. Bentuk dan isi transformasi ini ditentukan dari file XSL yang isinya berupa transformasi XML menggunakan XSLT (XSL Transformation).

Berikut ini beberapa kelebihan dan kelemahan Arsitektur Model 2X.

Tabel 2.5 Kelebihan dan Kelemahan Arsitektur Model 2X

Kelebihan	**Keterangan**
modular dan fleksibel	Style antarmuka dapat disentralisasi. Sebuah stylesheet dapat digunakan untuk keseluruhan *look and feel* dari aplikasi serta dapat digunakan lintas aplikasi, dan dapat diganti pada saat *runtime* sesuai kebutuhan.
Penggunaan Standar W3C	XSLT merupakan standar W3C dan telah diadopsi secara luas. Xalan, salah satu transformer XSLT telah dimasukkan ke dalam paket J2SE 1.4 dan kebanyakan aplikasi server sudah mendukung penggunaan XSLT.
Pipelining	Pada arsitektur yang lebih kompleks beberapa proses transformasi XSLT dapat disusun menjadi sebuah proses XML *pipeline*.
Mudah dipelajari bagi programmer JSTL (Java Server Pages™ Standard Tag Library)	Beberapa konstruksi bahasa JSTL mirip dengan XSLT. Sehingga dengan menguasai salah satunya akan mudah pula mempelajari yang lainnya.
Kelemahan	**Keterangan**
Adanya tambahan proses	Model 2X menambahkan proses transformasi tambahan pada layer presentasi.

Format XML pada JSP	JSP tidak memberikan validasi format XML pada hasil luaran, sehingga pengembang harus memastikan bahwa luaran JSP sesuai dengan format XML.

Pada bagian berikut dijelaskan komponen-komponen yang menyusun Arsitektur ini.

2.4.4.2. Komponen Controller

Komponen MVC controller melakukan pemetaan antara request yang datang ke operasi pada model aplikasi. Lalu controller mendelegasikan request ke komponen view tertentu berdasarkan model aplikasi dan state pengguna. Bagian ini memberikan beberapa aspek rancangan struktur dari controller.

Menentukan operasi yang akan dijalankan

Ketika controller menerima HTTP request dari client, ia harus dapat mengenali operasi yang sedang dilakukan pada aplikasi web. Sebagai contoh, dalam aplikasi menajemen pengguna, controller harus mengenali jenis request yang masuk, apakah membuat pengguna baru, menghapus pengguna, atau yang lainnya.

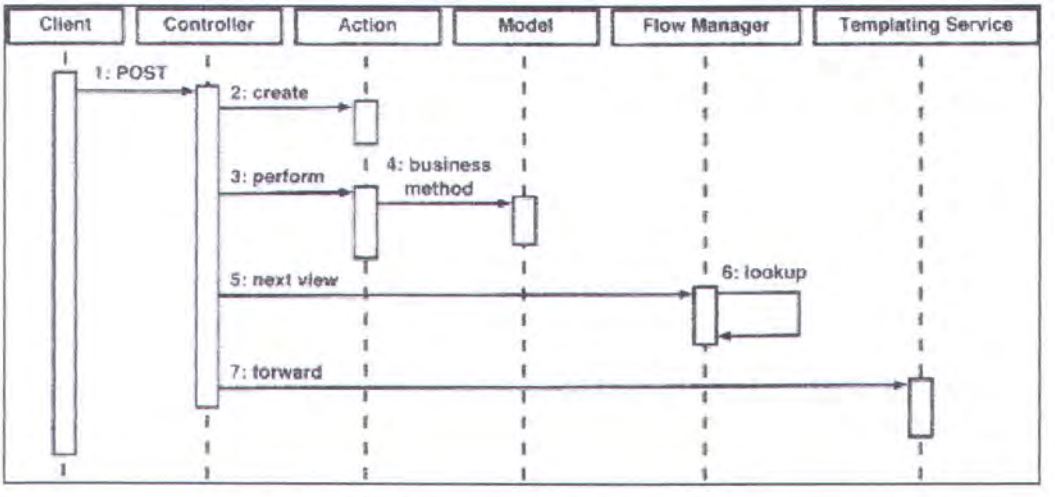
Sebagai contoh, untuk implementasi berupa servlet, dalam mengenali operasi ini dilakukan pemetaan servlet pada file deskriptor. Sehingga semua request yang memiliki akhiran tertentu akan masuk ke ke servlet ini. Operasi yang dijalankan diketahui dengan membaca URL yang masuk.

halaman yang lain. Sehingga urutan tampilan bergantung *link* yang ada pada halaman web tersebut. Sedangkan pada arsitektur MVC urutan tampilan bergantung hasil proses controller.

Biasanya tampilan berikutnya bergantung pada:

- Tampilan saat ini
- Hasil dari operasi yang dijalankan model
- Kondisi pada sisi server, yang berada pada PageContext, ServletRequest, HttpSession, and ServletContext.

Controller menggunakan data-data ini untuk menetukan tampilan apa yang diberikan selanjutnya. Gambar berikut adalah *sequence diagram* untuk sebuah request yang menunjukkan proses penentuan view.



Gambar 2.41. *Sequence Diagram* pada Controller

Berikut ini penjelasan gambar *sequence dagram* di atas :

1. Controller menerima request dari *client*.
2. Controller membuat sebuah objek `Action` sehubungan dengan operasi yang dibawa oleh request.
3. Controller memanggil method perform dari objek Action.

4. Method `perform` memanggil method proses bisnis dari model.
5. Controller memanggil *screen flow manager* untuk menetukan tampilan berikutnya.
6. *Screen flow manager* menentukan tampilan berikut dan mengambalikan namannya ke controller.
7. Controller meneruskan request ke *templating service*, yang menyusun *view* ke client.

2.4.4.3. Komponen View

Komponen View pada MVC menampilkan data yang dihasilkan oleh komponen model. Komponen view dapat berupa halaman JSP atau servlet. JSP sesuai untuk menampilkan tampilan yang berupa teks, seperti HTML atau XML. Sedangkan untuk menghasilkan *content* yang berbentuk data biner, Servlet lebih sesuai.

Browser HTML browsers merupakan *lightweight clients*, sehingga aplikasi web biasanya memberikan content yang dinamis dengan berbagai format tampilan. Untuk *client* yang berupa aplikasi *desktop* fungsionalitas view lebih banyak terimplementasi pada sisi *client*, dibandingkan pada sisi *server.*. *Client* seperti ini berupa aplikasi *stand-alone*, atau aplikasi yang memakali format khusus seperti MacroMedia Flash or Adobe Portable Document Format (PDF).

Aplikasi web tidak terbatas untuk diakses oleh browser HTML yang menggunakan protocol HTTP. Aplikasi web juga dapat diakses oleh *client* MIDP menggunakan protocol khusus, aplikasi yang menggunakan XML, atau web

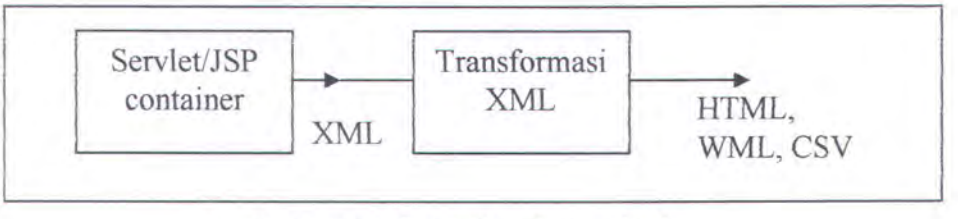
service yang mengirim request berupa pesan Electronic Business XML (ebXML) atau pesan Simple Object Access Protocol (SOAP). Dengan komponen view yang dirancang dengan baik, aplikasi web dapat manyatukan akses dari berbagai jenis *client* tersebut dengan komponen fungsional aplikasi yang sama.

2.4.4.4. Komponen Model

Komponen model merepresentasikan rancangan data aplikasi sekaligus *business logic* dari aplikasi. Desain komponen model MVC yang baik tidak berkaitan dengan teknologi tertentu atau layer aplikasi tertentu. Sehingga fleksibel dalam pemakaiannya. Pada bab 3.3.2 telah dibahas implementasi komponen Model pada aplikasi berbasis web.

2.4.4.5. Komponen Transformasi XML

Aspek terpenting dari Arsitektur Model 2X adalah penggunaan bahasa transformasi XML, seperti XSL Transformation (XSLT) dihubungkan dengan servlet dan JSP. Gambar berikut mengilustrasikan dasar dari Model 2X:



Gambar 2.42 Proses Transformasi XML

Luaran dari Servlet dan JSP biasanya dikirimkan langsung ke *browser*. Pada Model 2X luaran ini terlebih dahulu melalui proses transformasi XML sebelum dikirm ke *browser*. Kelemahan penggunaan JSP dalam menghasilkan luaran berformat XML adalah karena JSP digunakan untuk menghasilkan luaran

dalam berbagai format, sehingga JSP tidak melakukan validasi format XML terhadap luaran yang dikeluarkannya.

Setelah luaran XML dihasilkan oleh JSP, luaran ini siap dijadikan sebagai masukan alat transformasi XML. Salah satu alat yang dapat digunakan untuk proses transformasi ini adalah XSLT. Penjelasan mengenai XSLT dapat dilihat pada bab 2.1.2 tentang XSL. Hasil proses transformasi ini dapat berupa XML, HTML, atau sebarang format teks lainnya.

BAB III

PERANCANGAN SISTEM

BAB III

PERANCANGAN SISTEM

Bab ini berisi penjelasan proses perancangan aplikasi. Meliputi Deskripsi Kebutuhan Sistem, Perancangan Arsitektur Sistem, Perancangan Data, Perancangan Proses, serta Perancangan Antarmuka.

3.1. DESKRIPSI KEBUTUHAN SISTEM

Dalam deskripsi sistem ini akan dijelaskan tentang masukan dan keluaran dari sistem yang ada dan pembagian pengguna dari sistem termasuk akses-akses yang bisa dilakukan oleh masing-masing level pengguna.

Pada aplikasi ini yang menjadi data masukan adalah data informasi yang berkaitan dengan aplikasi ini, yaitu data mahasiswa, data dosen, arsip salinan lunak buku tugas akhir, dan artikel-artikel ilmiah. Data pengguna berupa pengguna mahasiswa dan dosen dimasukan oleh administrator web ke dalam database. Mahasiswa dan dosen dapat memasukkan data arsip sesuai hak aksesnya. Dan data-data ini akan diproses sehingga menghasilkan keluaran berupa informasi yang sudah diolah berdasarkan permintaan umum dari pengguna aplikasi ini.

Pengguna dari aplikasi sistem pengarsipan tugas akhir ini dapat dibagi menjadi 3 (tiga) jenis yaitu admin, mahasiswa dan dosen. Masing-masing jenis pengguna ini mempunyai hak akses yang berbeda sesuai dengan fungsinya masing-masing.

a. Administrator

Administrator adalah pengguna yang mempunyai hak penuh dalam mengakses aplikasi web ini, sehingga untuk itu diperlukan login terlebih dahulu sebelum mendapatkan fasilitas-fasilitas administrator. Hal-hal yang dapat dilakukan oleh administrator web pada aplikasi ini adalah :

- Memasukan data pengguna, yaitu mahasiswa dan dosen.
- Memasukkan data arsip secara keseluruhan, baik arsip tugas akhir maupun arsip artikel ilmiah.
- Memasukkan data-data pendukung aplikasi ini, misalnya bidang minat yang ada, jenis artikel yang disimpan, serta periode ujian tugas akhir.

b. Mahasiswa

Pengguna mahasiswa adalah mahasiswa yang sedang mengerjakan tugas akhir, ataupun telah menyelesaikan tugas akhir. Hal-hal yang dapat dilakukan oleh mahasiswa pada aplikasi ini adalah :

- Memasukkan data arsip tugas akhirnya, termasuk salinan lunak buku tugas akhir.
- Memasukkan data diri di aplikasi ini.
- Melakukan penelusuran terhadap data-data tugas akhir dan artikel ilmiah yang terdaftar di aplikasi ini.

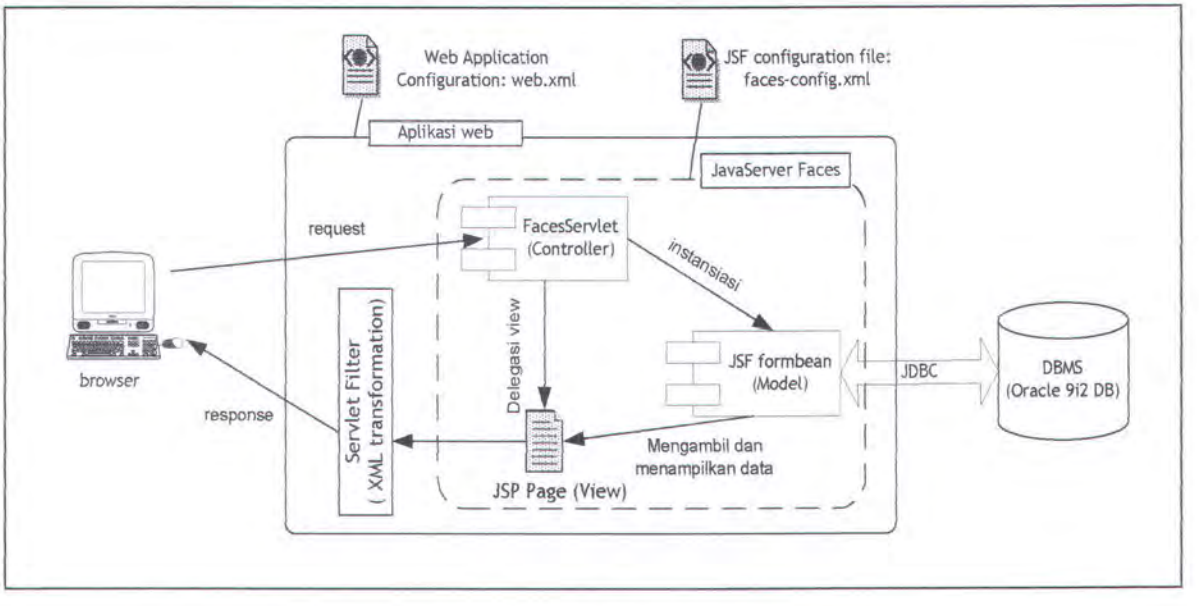
c. Dosen

Pengguna dosen adalah dosen yang terdaftar di universitas baik yang sedang membimbing mahasiswa tugas akhir maupun yang tidak. Hal-hal yang dapat dilakukan oleh pengguna dosen pada aplikasi ini adalah :

- Memasukkan data artikel ilmiah yang nantinya dapat dijadikan referensi bagi mahasiswa dalam mengerjakan tugas akhir.
- Memasukkan data diri di aplikasi ini.
- Melakukan penelusuran terhadap data-data tugas akhir dan artikel ilmiah yang terdaftar di aplikasi ini.

3.2. PERANCANGAN ARSITEKTUR SISTEM

Sub bab ini menjelaskan perancangan Arsitektur sistem. Sistem Pengarsipan Tugas Akhir ini menerapkan arsitektur Model 2X dengan framework JavaServer Faces sebagai struktur MVC-nya. Gambar 3.1 berikut memberikan skema arsitektur sistem secara keseluruhan.



Gambar 3.1 Arsitektur sistem

Sebagaimana yang telah dijelaskan pada bab 4.2.2 bahwa Arsitektur MVC terdiri dari 3 (tiga) buah komponen: Model, View dan Controller. Dalam

implementasinya, framework JSF telah menyediakan struktur Model-View-Controller ini yaitu:

- Model, dalam bentuk *formbean* berisi data dan proses bisnis sistem.
- View sebagai presentation layer berbentuk halaman JSP. Di sini juga diterapkan fitur-fitur JSF untuk menampilkan kontrol-kontrol HTML
- Controller, bertugas mengatur jalannya aplikasi. Dalam implementasinya, yang berperan sebagai controller adalah servlet. Servlet ini menerima request dari client, memanggil *Model* yang akan menangani request, dan meneruskan request ke *View*. Untuk mengarahkan semua request yang masuk, pada file deskriptor `web.xml` diberikan *pattern mapping* berupa path dengan awalan faces. Dengan cara ini semua request yang memiliki bagian path "/faces" harus melalui Faces servlet dari JSF.

```
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

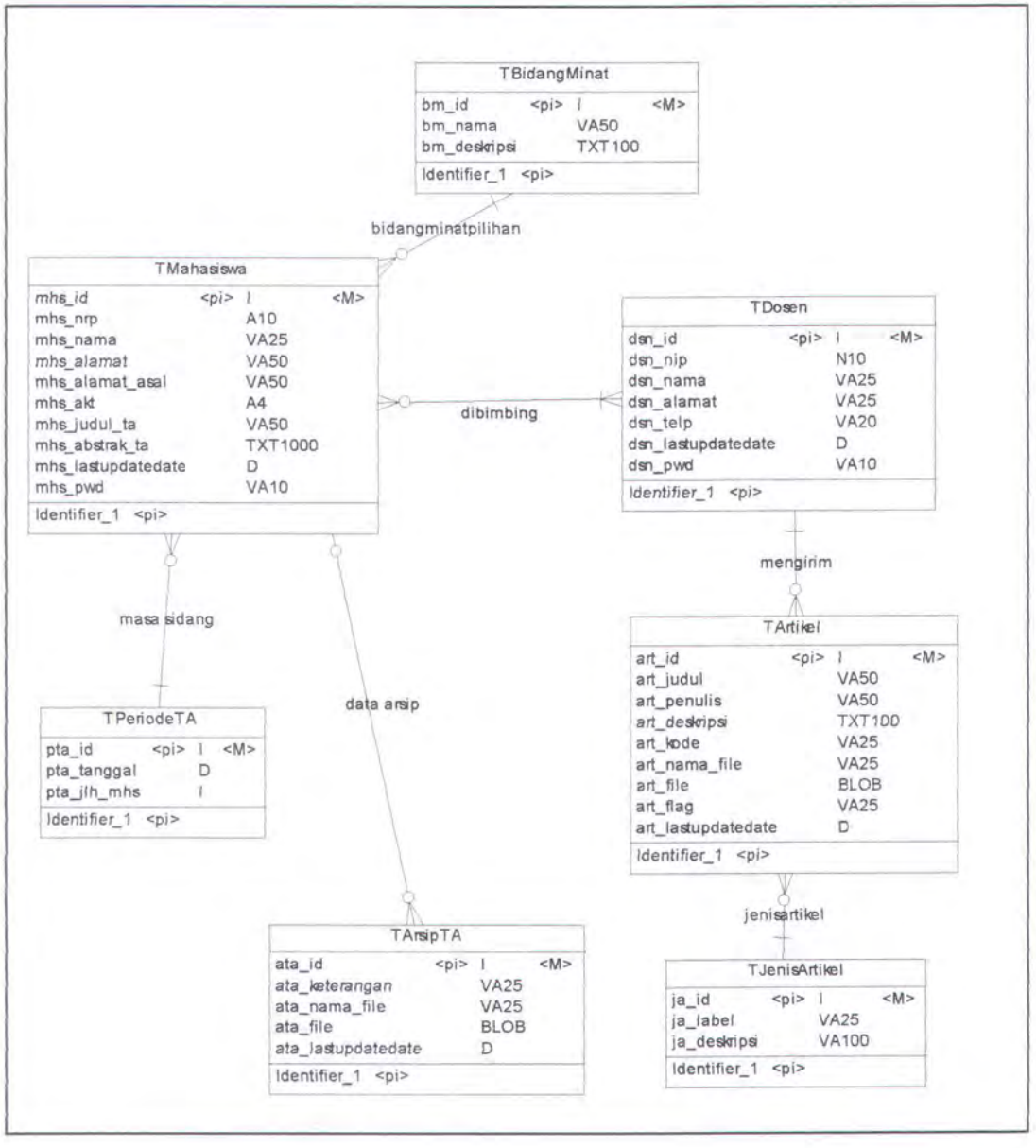
Gambar 3.2 Servlet mapping untuk FacesServlet JSF

Pada Arsitektur Model 2X, keluaran dari bagian MVC ditransformasi terlebih dahulu melalui proses transformasi XML sebelum dikirim ke *client*. Dengan demikian keluaran dari bagian MVC harus memenuhi format XML.

Implementasi dari komponen transformasi ini berupa servlet filter. Servlet filter menerima hasil proses JSP/JSF yang sudah berformat XML. Transformasi XML mengolah keluaran yang diterima menjadi tampilan HTML. Definisi transformasi yang dilakukan berupa file XSL.

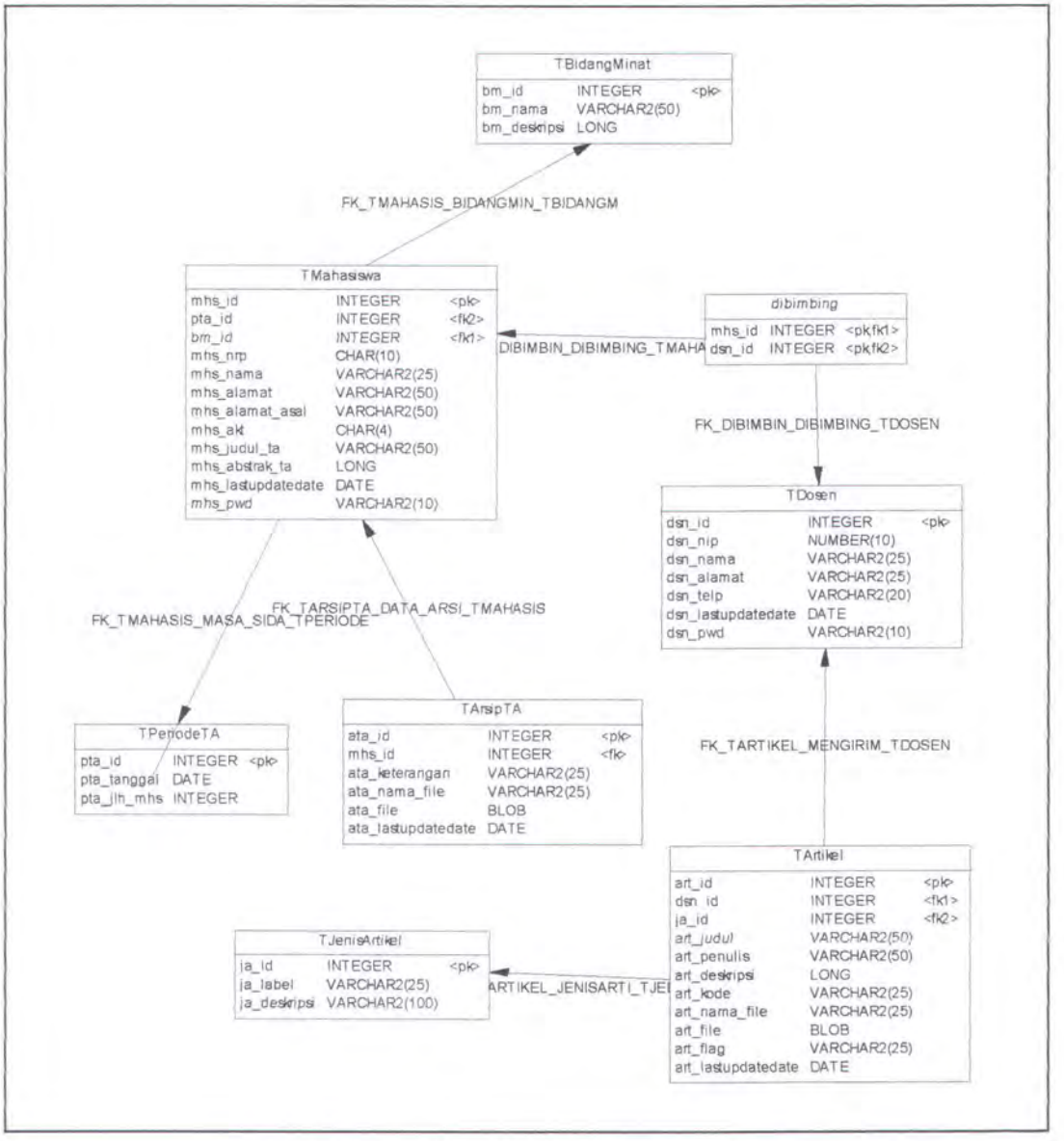
3.3 PERANCANGAN DATA

Dalam bagian ini akan dijelaskan mengenai perancangan basis data perangkat lunak yang akan dibuat dengan menggunakan *Power Designer*. Gambar 3.3 menggambarkan rancangan *conceptual data model* yang dipakai pada aplikasi:



Gambar 3.3 Conceptual Data Model Aplikasi Pengarsipan Tugas Akhir

Physical Data Model yang dihasilkan dapat dilihat pada gambar 3.4 berikut. Pada bagian ini juga dijelaskan tiap-tiap tabel secara rinci.



Gambar 3.4 Physical Data Model Aplikasi Pengarsipan Tugas Akhir

3.3.1. Tabel TMahasiswa.

Tabel TMahasiswa berisi data semua mahasiswa yang berhak mengakses aplikasi Pengarsipan Tugas Akhir ini berikut data tentang Tugas Akhir yang diambil mahasiswa. Tabel ini tidak menggunakan NRP mahasiswa sebagai

Primary Key untuk menambah fleksibilitas sistem, misalnya menangani adanya konversi NRP.

Tabel 3.1 Penjelasan Tabel TMahasiswa.

No	Nama Field	Tipe Data	Keterangan
1	MHS_ID	INTEGER	ID pengguna Mahasiswa
2	PTA_ID	INTEGER	ID record arsip Tugas Akhir
3	BM_ID	INTEGER	ID Bidang minat mahasiswa
4	MHS_NRP	CHAR(10)	NRP Mahasiswa
5	MHS_NAMA	VARCHAR2(25)	Nama mahasiswa
6	MHS_ALAMAT	VARCHAR2(50)	Alamat mahasiswa
7	MHS_ALAMAT_ASAL	VARCHAR2(50)	Alamat asal mahasiswa
8	MHS_AKT	CHAR(4)	Tahun masuk mahasiswa
9	MHS_JUDUL_TA	VARCHAR2(50)	Judul Tugas Akhir mahasiswa
10	MHS_ABSTRAK_TA	LONG	Abstrak TA Mahasiswa
11	MHS_LASTUPDATEDATE	DATE	Tanggal terakhir data ini diperbaharui
12	MHS_PWD	VARCHAR2(10)	Password Mahasiswa

3.3.2. Tabel TArsipTA

Tabel ini berisi data file Arsip Tugas Akhir. File Arsip disimpan dalam bentuk BLOB. Seorang mahasiswa bisa saja memiliki beberapa file arsip di sini. Tabel TMahasiswa dengan tabel ini memiliki relasi One-to-Many.

Tabel 3.2 Penjelasan Tabel TarsipTA

No	Nama Field	Tipe Data	Keterangan
1	MHS_ID	INTEGER	Foreign Key dari tabel TMahasiwa berisi id mahasiswa pemilik file
2	ATA_KETERANGAN	VARCHAR2(25)	Keterangan dari file Arsip
3	ATA_NAMA_FILE	VARCHAR2(25)	Nama file Arsip.
4	ATA_FILE	BLOB	Berisi data file Arsip
5	ATA_LASTUPDATEDATE	DATE	Tanggal terakhir data ini diperbaharui
6	ATA_ID	NUMBER	ID file arsip yang diupload. berfungsi sebagai Primary Key untuk tabel ini.

3.3.3. Tabel TDosen

Tabel ini berisi data dosen yang memiliki hak akses ke sistem. Pada tabel ini NIP Dosen tidak digunakan sebagai Primary Key untuk menambah fleksibilitas sistem misalnya mengantisipasi kasus seperti dosen yang belum memiliki NIP.

Tabel 3.3 Penjelasan Tabel Tdosen

No	Nama field	Tipe Data	Keterangan
1	DSN_ID	INTEGER	ID dosen yang bersangkutan, berfungsi sbg Primary Key untuk tabel ini
2	DSN_NIP	CHAR(10)	Berisi Nomor NIP Dosen yang

			bersangkutan
3	DSN_NAMA	VARCHAR2(25)	Nama Dosen yang bersangkutan
4	DSN_ALAMAT	VARCHAR2(25)	Alamat dosen yang bersangkutan
5	DSN_TELP	VARCHAR2(20)	Nomor Telepon dosen yang bersangkutan
6	DSN_LASTUPDATEDATE	DATE	Tanggal terakhir data ini diperbaharui
7	DSN_PWD	VARCHAR2(10)	Berisi data password dosen

3.3.4. Tabel TArtikel

Tabel ini berisi Data Artikel yang di-*upload* oleh dosen.

Tabel 3.4 Penjelasan Tabel TArtikel

No.	Nama Field	Tipe Data	Keterangan
1	ART_ID	INTEGER	ID data Artikel yang di-*upload*. Berfungsi sebagai Primary Key
2	DSN_ID	INTEGER	ID Dosen yang meng-*upload* artikel
3	JA_ID	INTEGER	ID Jenis Artikel, merupakan foreign key dari tabel TJenisArtikel
4	ART_JUDUL	VARCHAR2(70)	Judul artikel
5	ART_PENULIS	VARCHAR2(70)	Penulis Artikel
6	ART_DESKRIPSI	LONG	Deskripsi Artikel
7	ART_TANGGAL_TERBIT	DATE	Tanggal Publikasi artikel
8	ART_NAMA_FILE	VARCHAR2(70)	Nama file artikel

9	ART_FILE	BLOB	Data file artikel
10	ART_FLAG	VARCHAR2(25)	Penanda apakah file ini turut diindeks atau tidak
11	ART_LASTUPDATEDATE	DATE	Tanggal terakhir data ini diperbaharui

3.3.5. Tabel dibimbing

Tabel merupakan tabel hasil relasi Many-to-Many antara tabel TMahasiswa dengan tabel TDosen. Relasi ini merepresentasikan hubungan antara Mahasiswa Tugas Akhir dengan Dosen Pembimbingnya.

Tabel 3.5 Penjelasan Tabel Relasi dibimbing.

No	Nama Field	Tipe Data	Keterangan
1	MHS_ID	INTEGER	ID Mahasiswa Tugas Akhir
2	DSN_ID	INTEGER	ID Dosen Pembimbing

3.3.6. Tabel TBidangMinat

Tabel ini berisi daftar bidang minat yang dapat diambil oleh Mahasiswa. Tabel ini direferensikan oleh tabel TMahasiswa.

Tabel 3.6 Penjelasan Tabel TBidangMinat.

No	Nama Field	Tipe Data	Keterangan
1	BM_ID	INTEGER	ID Bidang Minat, berfungsi sebagai Primary Key
2	BM_NAMA	VARCHAR2(50)	Nama Bidang Minat
3	BM_DESKRIPSI	LONG	Penjelasan tentang Bidang Minat

3.3.7. Tabel TPeriodeTA

Tabel ini menyimpan data waktu Ujian Sidang Tugas Akhir. Tabel ini direferensikan oleh tabel TMahasiswa untuk menyimpan data tentang kapan mahasiswa tersebut mengikuti Ujian Sidang.

Tabel 3.7 Penjelasan Tabel TperiodeTA.

No	Nama Field	Tipe Data	Keterangan
1	PTA_ID	INTEGER	ID Waktu Ujian, berfungsi sebagai Primary Key
2	PTA_TANGGAL	DATE	Tanggal pelaksanaan Ujian Sidang
3	PTA_JLH_MHS	INTEGER	Jumlah Mahasiswa yang mengikuti Ujian Sidang Tugas Akhir

3.3.8. Tabel TJenisArtikel

Tabel ini menyimpan daftar jenis artikel yang disimpan di database Sistem Pengarsipan Tugas Akhir ini.

Tabel 3.8 Penjelasan Tabel TjenisArtikel

No	Nama Field	Tipe Data	Keterangan
1	JA_ID	INTEGER	ID jenis artikel, berfungsi sebagai Primary Key
2	JA_LABEL	VARCHAR2(25)	Nama Jenis Artikel
3	JA_DESKRIPSI	VARCHAR2(100)	Penjelasan tentang jenis artikel ini

3.3.9. Indeks Oracle Text

Pada Aplikasi ini digunakan indeks Oracle Text untuk membantu pencarian dalam dokumen teks. Tabel yang menyimpan dokumen teks yaitu tabel

TArsipTA (pada kolom ATA_FILE) dan tabel TArtikel(pada kolom ART_FILE). Karena ukuran dokumen yang akan disimpan relatif besar, maka pada kedua tabel ini dibuatkan indeks Oracle Text dengan jenis CONTEXT.

3.4. PERANCANGAN PROSES

Pada bagian ini dijelaskan rancangan sistem Pengarsipan Tugas Akhir yang nantinya akan diimplementasikan dalam bentuk aplikasi berbasis web. Proses perancangan ini akan digambarkan menggunakan notasi UML. Diagram UML yang dipakai adalah: Use Case Diagram, Activity Diagram, Class Diagram, serta Collaboration Diagram.

3.4.1. Pembuatan Use Case View

Bagian ini bertujuan untuk memberikan gambaran bagaimana tingkah laku sistem. Untuk membuat Use Case View, diawali dengan menentukan Actor dari sistem, kemudian menentukan Use Case dari sistem, dan terakhir membuat Activity Diagram untuk Use Case yang membutuhkan penjelasan lebih detail.

3.4.1.1. Actor

Aktor merepresentasikan orang atau objek yang akan berinteraksi dengan sistem. Untuk sistem Pengarsipan Tugas Akhir ini didefinisikan 3 (tiga) aktor yang akan berinteraksi dengan sistem, yaitu Administrator, Mahasiswa, dan Dosen. Pada tabel berikut dijelaskan definisi dari tiap aktor.

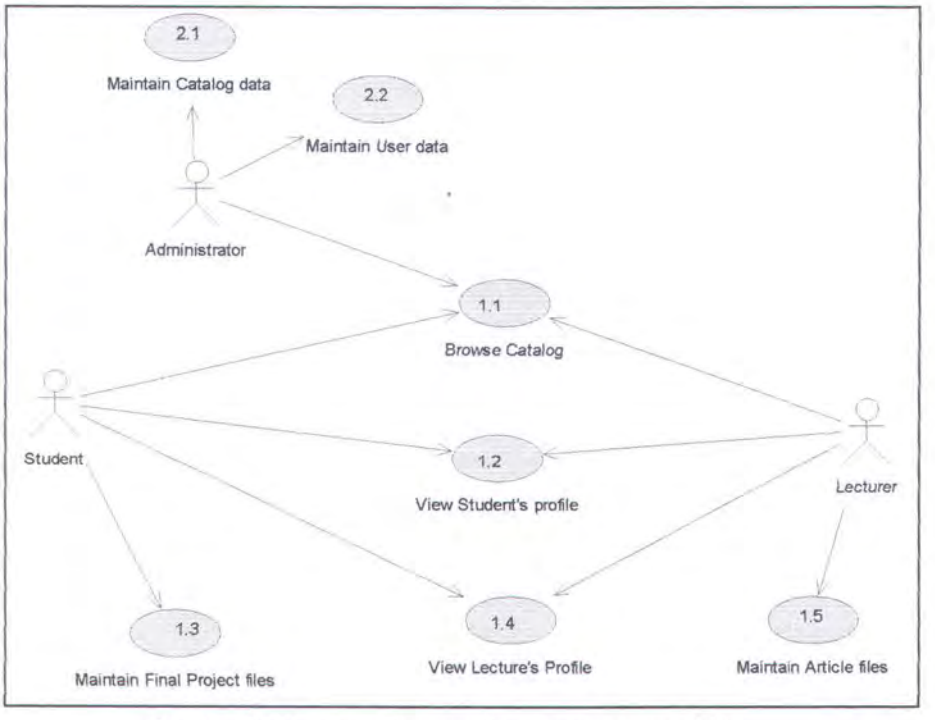
Tabel 3.9 Aktor dan definisinya dalam sistem.

No	Nama Aktor	Definisi
1	Administrator	Orang yang mengatur setting aplikasi secara keseluruhan, bertanggung jawab terhadap administrasi

		data arsip serta data *pengguna*.
2	Mahasiswa	Orang yang terdaftar di universitas sebagai mahasiswa dan telah mengambil mata kuliah Tugas Akhir. Mahasiswa dapat mengisi biodatanya serta meng-*upload* arsip Tugas Akhirnya di database.
3	Dosen	Orang yang mengajar di universitas. Dosen dapat mengisi biodatanya dan meng-*upload* Artikel yang berhubungan dengan perkuliahan.

3.4.1.2. Use Cases

Use Case adalah representasi dialog antara aktor dengan sistem, berupa urutan transaksi yang dilakukan oleh sistem dan aktor. Pada gambar berikut diberikan diagram use case secara keseluruhan.



Gambar 3.5 Use Case Diagram Pengarsipan tugas akhir

- Use Case 1.1. Browse Catalog.

Use Case in berupa aktivitas penelusuran data katalog. Aktivitas penelusuran dapat berupa penelusuran langsung ke daftar Tugas Akhir dan Daftar artikel. Aktivitas penelusuran dapat pula berupa pencarian melalui kata kunci tertentu. Aktivitas ini dapat dilakukan oleh ketiga aktor.

- Use Case 1.2. View Student profile.

Use Case ini berupa aktivitas melihat biodata mahasiswa, dan khusus untuk mahasiswa, ia dapat melakukan perubahan biodatanya. Pada biodata ini juga terdapat *link* ke biodata dosen pembimbing.

- Use Case 1.3 Maintain Final Project files.

Mahasiswa dapat memeriksa daftar file yang telah di-*upload* ke database. Ia juga dapat menambah file, men-*download* file, atau menghapus file yang telah di-*upload.*

- Use Case 1.4 View Lecturer profile.

Melihat biodata Dosen. Untuk *Actor* selain dosen aktivitas ini hanya untuk melihat informasi biodata Dosen. Untuk *Actor* Dosen sendiri, ia dapat melakukan perubahan biodata dirinya.

- Use Case 1.5 Maintain Article files.

Dosen dapat memeriksa daftar file artikel yang telah di-*upload* ke database. Ia juga dapat menambah artikel, men-*download* artikel, atau menghapus artikel yang telah di-*upload.*

- Use Case 2.1 Maintain Catalog data.

Use Case ini berupa aktivitas administrator untuk mengatur data katalog sistem, yang terdiri dari data Arsip Tugas Akhir dan data file Artikel. Administrator dapat melihat daftar file di database, menambah file, men-*download* file, atau menghapus file yang ada di database.

- Use Case 2.2. Maintain User data.

Use Case ini berupa aktivitas administrator untuk mengatur data pengguna sistem, yang terdiri dari data mahasiswa dan data dosenl. Administrator dapat melihat daftar pengguna di database, menambah pengguna, melakukan perubahan data pengguna, atau menghapus pengguna yang ada di database.

3.4.1.3. Activity Diagram

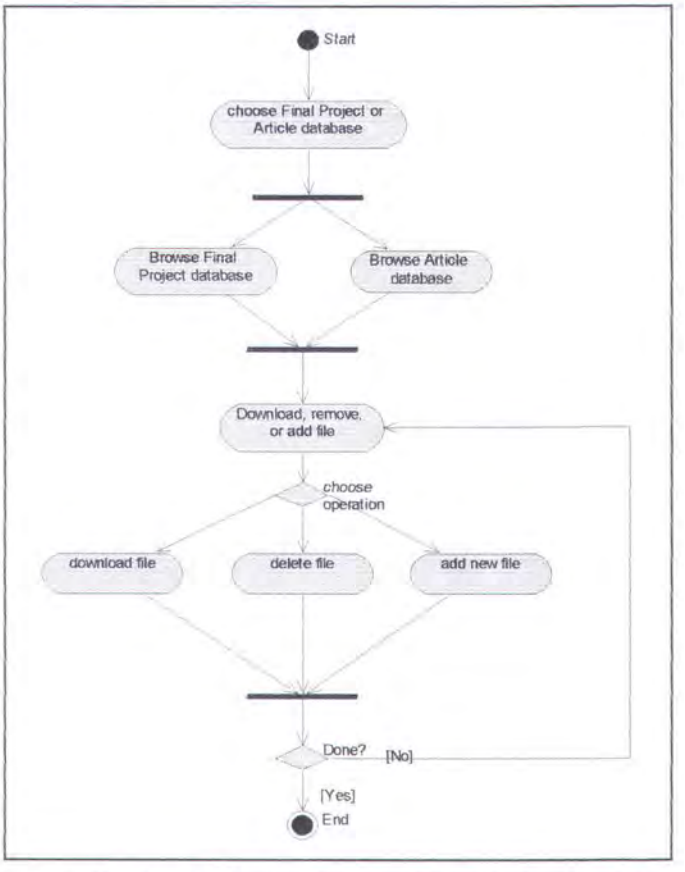
Untuk merepresentasikan pelaksanaan fungsi dari use case dalam bentuk alur kerja, dibuat *Activity Diagram.* Use case yang dibuatkan *Activity Diagram*-nya adalah use case yang merupakan proses inti dari aplikasi.

Pada bagian ini akan diberikan *Activity Diagram* untuk use case 2.1. Maintain Catalog Data, Browse Catalog, dan View Student profile.

Maintain Catalog Data

Lihat *Activity Diagram* pada gambar 3.6. Diagram ini menunjukkan alur kerja administrator ketika mengatur data katalog. Aktivitas ini dimulai dengan memilih data katalog yang akan diatur, yaitu data Arsip Tugas Akhir atau data Artikel. Kemudian administrator dapat melihat tabel daftar file yang akan diatur. Pada tabel ini administrator dapat memilih operasi untuk setiap file. Operasi

tersebut meliputi mendownload file, menghapus file dari database, atau menambah/meng-*upload* file.

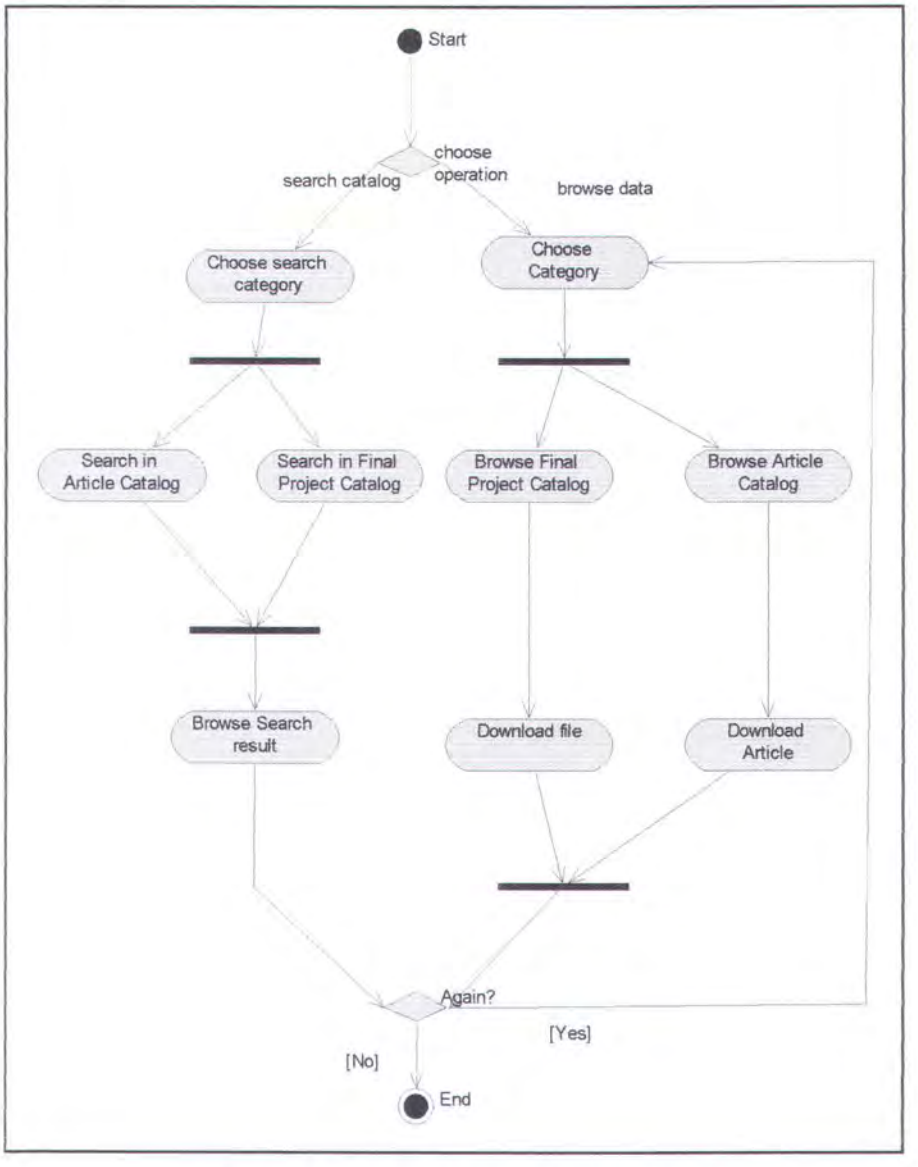


Gambar 3.6. Activity Diagram Maintain Catalog data

Browse Catalog

Lihat *Activity Diagram* pada gambar 3.7. Diagram ini menunjukkan alur kerja *Actor* yang melakukan penelusuran katalog. Aktivitas ini dimulai dengan memilih cara penelusuran. Ada dua cara yang dapat dipilih, yaitu melalui pencarian dan melalui penelusuran data tabular. Jika *Actor* memilih cara yang pertama, pengguna mengisi *form* pencarian. Setelah diperoleh hasil pencarian, *Actor* dapat menelusuri hasil pencarian tersebut. Untuk cara yang kedua, *Actor* memilih untuk menelusuri data Arsip Tugas Akhir atau data Artikel dari dosen.

Dengan cara ini *Actor* memperoleh daftar file Tugas Akhir atau Artikel dan dapat di download.

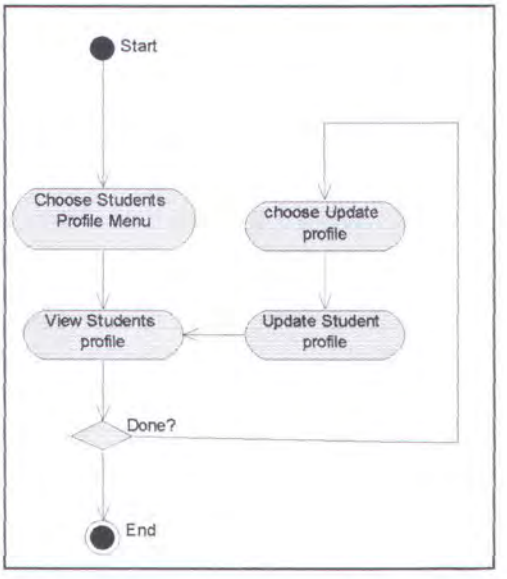


Gambar 3.7. Activity Diagram Browse Catalog.

View Student's Profile

Activity Diagram View Student's Profile digambarkan pada gambar 3.8. Aktivitas ini dimulai dengan pemilihan menu melihat profil mahasiswa. Selanjutnya *Actor* dapat merubah biodata dengan memilih menu edit data. Form

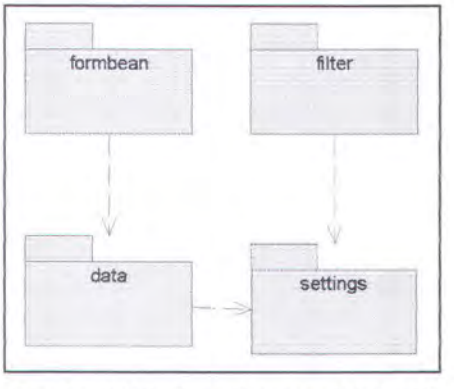
Edit Profil ditampilkan. Setelah *Actor* memperbaharui data, *Actor* menyimpan perubahan dengan memilih menu "simpan data".



Gambar 3.8 Activity Diagram View Student's Profile

3.4.2. Class Diagram

Sub bab ini menjelaskan kelas-kelas yang dibuat untuk menyusun arsitektur Model 2X. Kelas-kelas dibagi ke dalam sejumlah *package* untuk mengelompokkan kelas-kelas tersebut sesuai fungsinya (lihat tabel 3.10). Pada gambar 3.9 diberikan *Package Diagram* kelas-kelas pada aplikasi ini.



Gambar 3.9 Package Diagram Model 2X

Tabel 3.10 Pengelompokan kelas ke dalam empat package

Package	Keterangan
Formbean	Kelas-kelas di *package* ini berperan sebagai data model. Misalnya kelas MahasiswaBean dan DosenBean yang memodelkan Mahasiswa dan Dosen.
Data	Menangani koneksi ke database serta perintah-perintah dasar akses data ke database.
Filter	Berperan dalam proses transformasi XML pada Presentation Layer.
Settings	Berperan menangani konfigurasi aplikasi: akses ke database dan urutan transformasi XML.

Di bagian ini akan diberikan beberapa *Class Diagram* untuk menjelaskan kelas-kelas yang akan menyusun Arsitektur Model 2X. Ada tiga buah diagram yang akan digambarkan, yaitu *Class Diagram* Model Component, Filter Component, serta Application Settings.

3.4.2.1. Model Component

Pada bagian ini digambarkan kelas-kelas yang terdapat pada *package* data dan *package* formbean. Kedua *package* ini digambarkan dalam satu diagram karena eratnya keterkaitan antara kedua *package*.

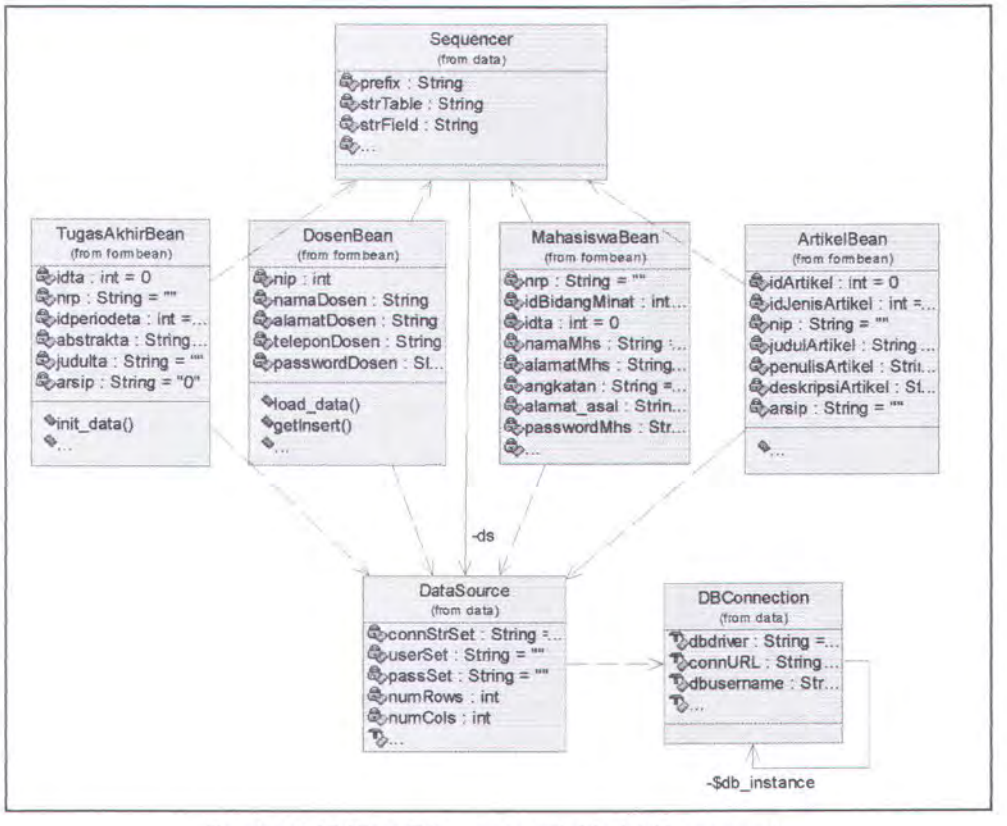
Dengan penggunaan JavaServer Faces, bentuk MVC sudah terwujud dari komponen-komponen JSF itu sendiri. Bagian Model dipenuhi dari *formbean* yang merepresentasikan model data. Bagian View berupa file JSP dimana pengembang menyusun antarmuka halaman web. Sedangkan bagian Controller adalah FacesServlet yang menerima semua request sebelum diteruskan ke komponen JSF

lainnya. Dengan demikian hanya bagian model yang digambarkan Class Diagramnya.

Pada diagram (lihat Gambar 3.10) di bawah ada empat buah kelas yang berfungsi sebagai model, yaitu:

- ArsipTABean, memodelkan Tugas Akhir mahasiswa
- DosenBean, memodelkan Dosen pengajar.
- MahasiswaBean, memodelkan Mahasiswa.
- ArtikelBean, memodelkan Artikel yang dikirim oleh dosen.

Kelas DosenBean dan MahasiswaBean dipakai sebagai *formbean* JSF untuk halaman yang menangani pengolahan data dosen dan mahasiswa. Sedangkan kelas ArsipTABean dan ArtikelBean digunakan sebagai javabeans biasa, karena dipakai dalam proses *upload* dan *download file*.



Gambar 3.10 Class Diagram untuk Model Component

Kelas DataSource dan DBConnection adalah bagian dari Package data. Pada Diagram ini digambarkan Kelas-kelas model memiliki hubungan dependent (garis panah putus-putus) terhadap kelas di package Data. Untuk kelas MahasiswaBean dan DosenBean terdapat inner class Action JSF untuk melakukan operasi-operasi yang berkaitan dengan perubahan data. Pada inner class inilah dilakukan proses bisnis, di sini juga digunakan kelas DataSource, sehingga kelas-kelas Model memiliki relasi dependent terhadap DataSource.

3.4.2.2. Filter Component

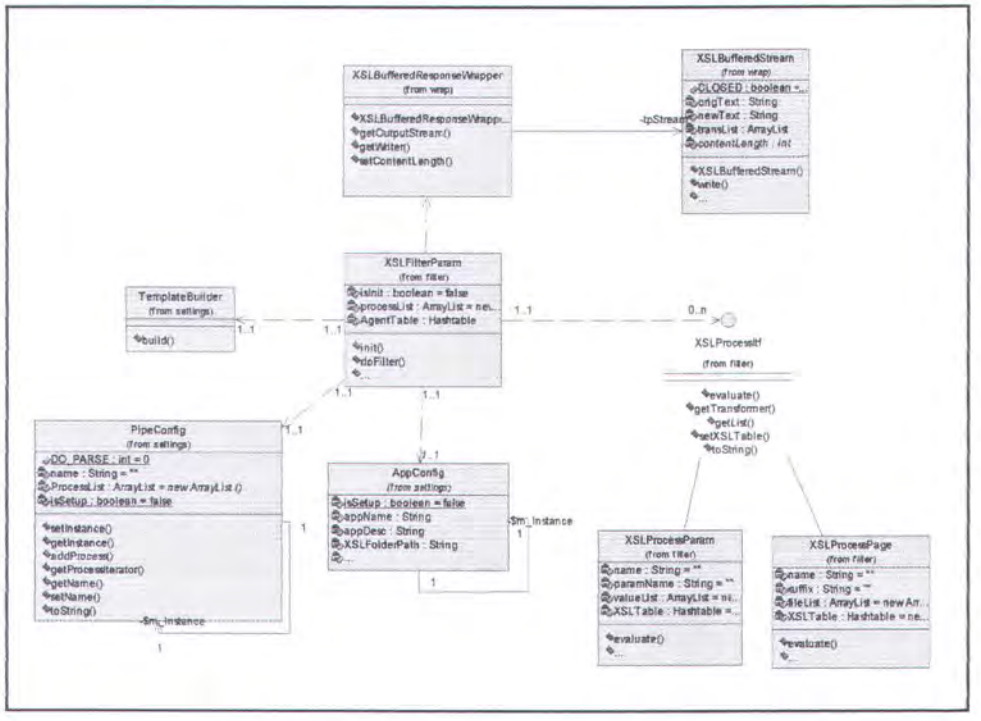
Pada Class Diagram ini (lihat Gambar 3.11) digambarkan kelas-kelas yang menyusun komponen filter yang berfungsi melakukan proses Transformasi XML. Kelas-kelas ini berasal dari *package* filter. Adapun kelas-kelas tersebut adalah sebagai berikut:

- XSLFilterParam

  Kelas ini adalah Servlet Filter sebagai tempat melakukan filtering terhadap request yang masuk ataupun keluar. Kelas ini menyimpan ArrayList dari XSLProcessItf yang merupakan elemen proses transformasi XML.

- XSLProcessItf

  Adalah Interface dari proses transformasi XML terhadap response yang akan keluar dari aplikasi web. Interface ini memiliki dua method penting, yaitu: evaluate() dan getTransformer(). Method evaluate() digunakan untuk memeriksa variable yang menjadi parameter aktif tidak nya proses transformasi yang bersangkutan.



Gambar 3.11 Class Diagram untuk Filter Component

- XSLProcessParam

Adalah salah satu kelas implementasi dari XSLProcessItf. Kelas ini mempunyai karakteristik bahwa proses transformasi yang mana yang dilakukan tergantung nilai dari parameter tertentu di variabel session. Sehingga tidak tergantung nama file yang diakses oleh client.

- XSLProcessPage

Merupakan kelas implementasi XSLProcessItf dengan karakteristik pemilihan proses transformasi tergantung dari nama file yang diakses dan nilai dari veriabel tertentu pada session.

- XSLBufferedResponseWrapper

Kelas ini berfungsi untuk membungkus response yang berasal dari *web container*. Objek response dibungkus agar proses transformasi dapat dilakukan.

Dengan cara ini, *web container* tidak akan menulis data langsung ke layer, namun ditampung terlebih dahulu di buffer response wrapper ini.

- XSLBufferedStream

Hampir sama dengan XSLBufferedResponseWrapper, tetapi dalam hal ini yang dibungkus adalah ServletOutputStream. Pada kelas ini ditampung Transformer yang diperoleh dari XSLProcessItf. Selain menapung data dari *web container*, pada kelas ini juga dilakukan proses transformasi XML yang sesungguhnya dengan menggunakan Transformer yang disimpan.

- TemplateBuilder

Kelas ini bertugas menghasilkan Hashtable yang berisi Template dari file XSL yang telah di-*parsing*. Hanya ada satu method yang dimiliki kelas ini, yaitu build().Masukan yang diperlukan adalah list nama file.

- AppConfig dan PipeConfig (*package* setting)

Kedua kelas ini adalah bagian dari package Setting. Berguna untuk mengambil nilai-nilai dari parameter setting aplikasi.Kelas AppConfig menangani setting aplikasi secara umum, contohnya koneksi database. PipeConfig menangani setting pipeline proses transformasi XML.

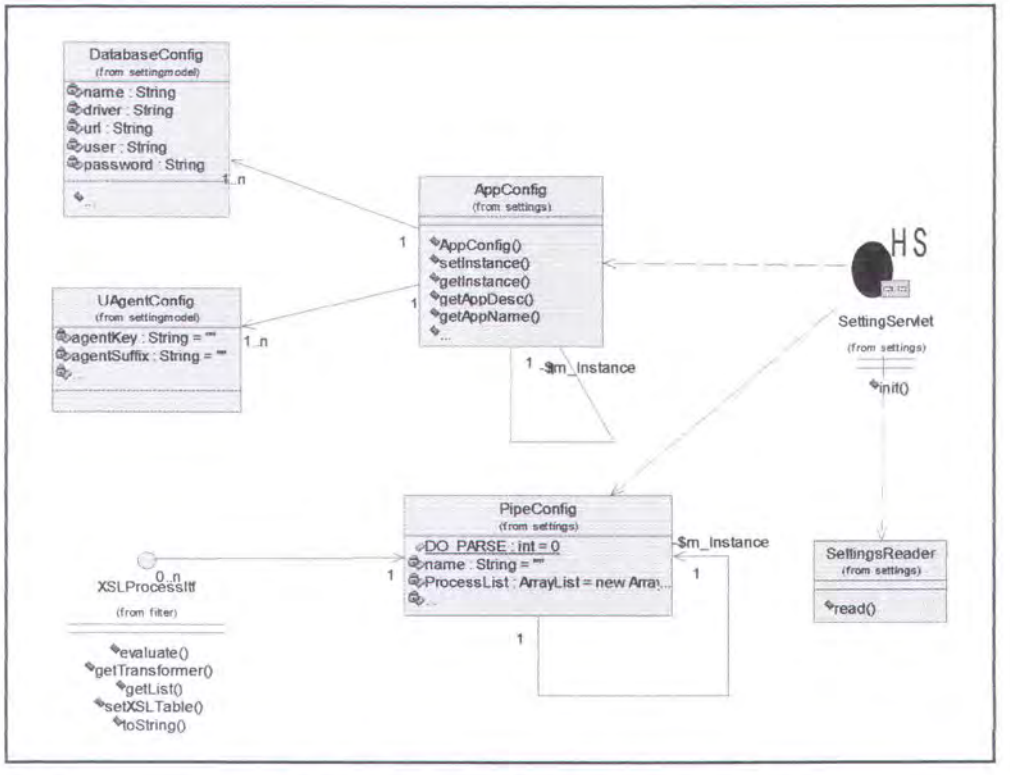
3.4.2.3. Application Configuration

Class Diagram Application Settings menggambarkan kelas-kelas yang digunakan untuk menyediakan informasi konfigurasi aplikasi bagi kelas-kelas atau komponen-komponen aplikasi yang membutuhkan sejumlah nilai konfigurasi. Nilai-nilai ini nantinya diisikan pada file konfigurasi. Kelas-kelas ini berasal dari *package* setting.

Berikut penjelasan tiap kelas-kelas pada *class diagram* gambar 3.12.

- AppConfig.

Kelas ini berisi konfigurasi aplikasi secara umum. Konfigurasi tersebut adalah konfigurasi database, username dan password administrator untuk aplikasi ini, serta konfigurasi servlet yang digunakan untuk mendeteksi tipe *device* dari *client*.



Gambar 3.12 Class Diagram untuk Application Settings.

- DatabaseConfig.

Kelas ini berisi konfigurasi untuk akses ke database, meliputi *username*, *password*, alamat *URL* database, serta nama kelas *driver* yang dipakai. Karena konfigurasi di aplikasi ini memungkinkan untuk menyimpan konfigurasi beberapa database, maka nantinya pada saat *run-time* objek ini akan dibangkitkan sejumlah konfigurasi database yang ada. Objek-objek ini disimpan sebagai *member variable* dari objek AppConfig.

- PipeConfig.

Kelas ini berisi konfigurasi *XML Pipeline* aplikasi. Pada kelas ini disimpan sebuah array dari XSLProcessItf sebagai satuan proses pengolahan XML.

- SettingServlet.

Kelas ini adalah servlet yang bertugas membaca file konfigurasi lalu membangkitkan objek AppConfig dan PipeConfig berdasarkan isi file konfigurasi tersebut. Untuk proses pembacaan file konfigurasi, digunakan kelas SettingsReader.

- UAgentConfig

Kelas ini digunakan untuk menyimpan data tentang jenis perangkat *client* yang didukung oleh aplikasi ini.

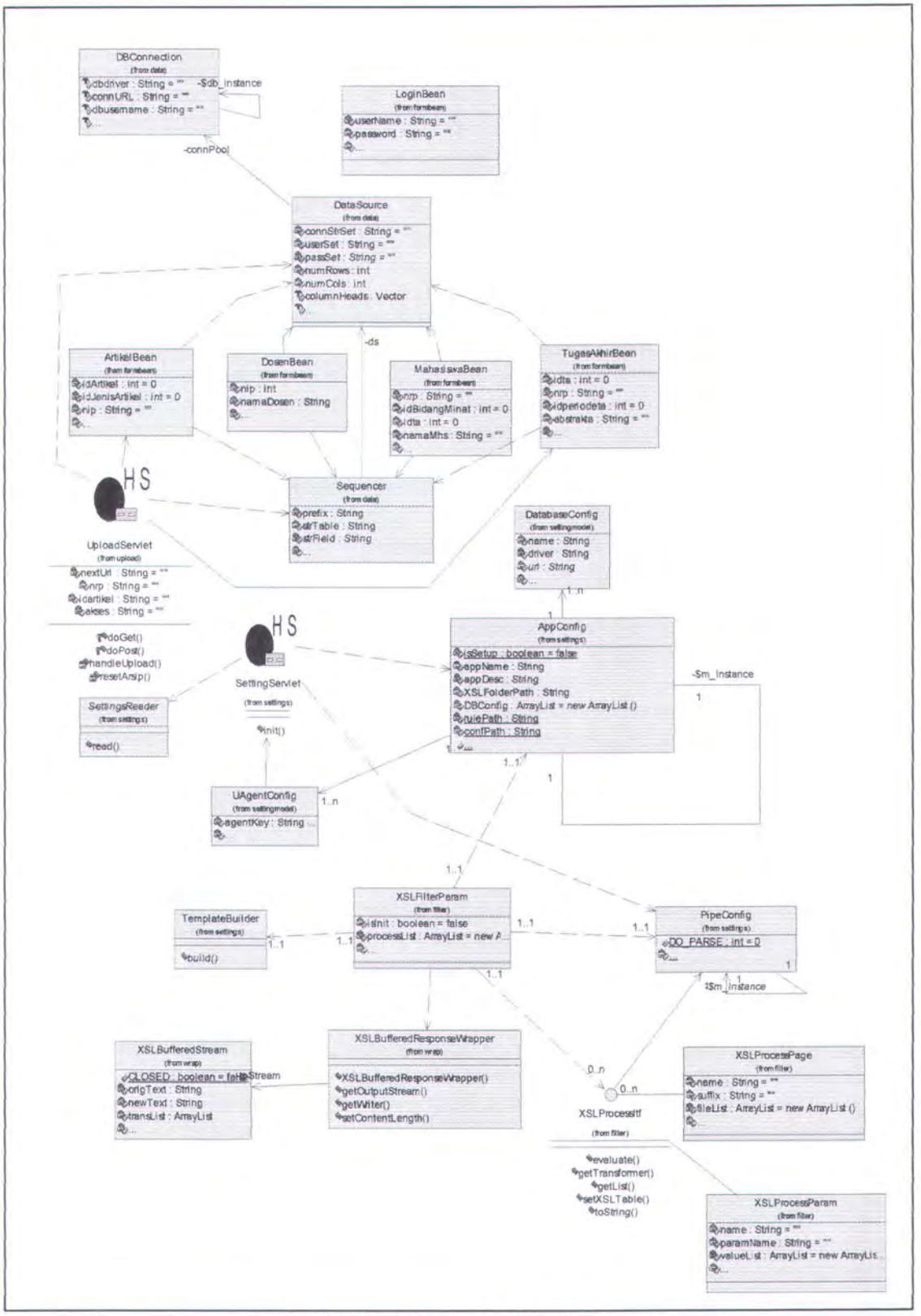
- SettingsReader

Kelas ini digunakan oleh SettingServlet untuk membaca file konfigurasi. Untuk fungsi pembacaan, kelas ini memiliki sebuah method read() yang memiliki argumen nama file yang akan dibaca. Hasil pembacaan berupa objek AppConfig atau PipeConfig, sesuai dengan isi file yang dibaca.

- XSLProcessItf (*package* filter)

Pada kelas PipeConfig terdapat sebuah array dari interface XSLProcessItf. Isi dari array ini nantinya dipakai untuk melakukan proses transformasi XML.

Class diagram pada halaman berikut memperlihatkan hubungan antar kelas-kelas secara keseluruhan.



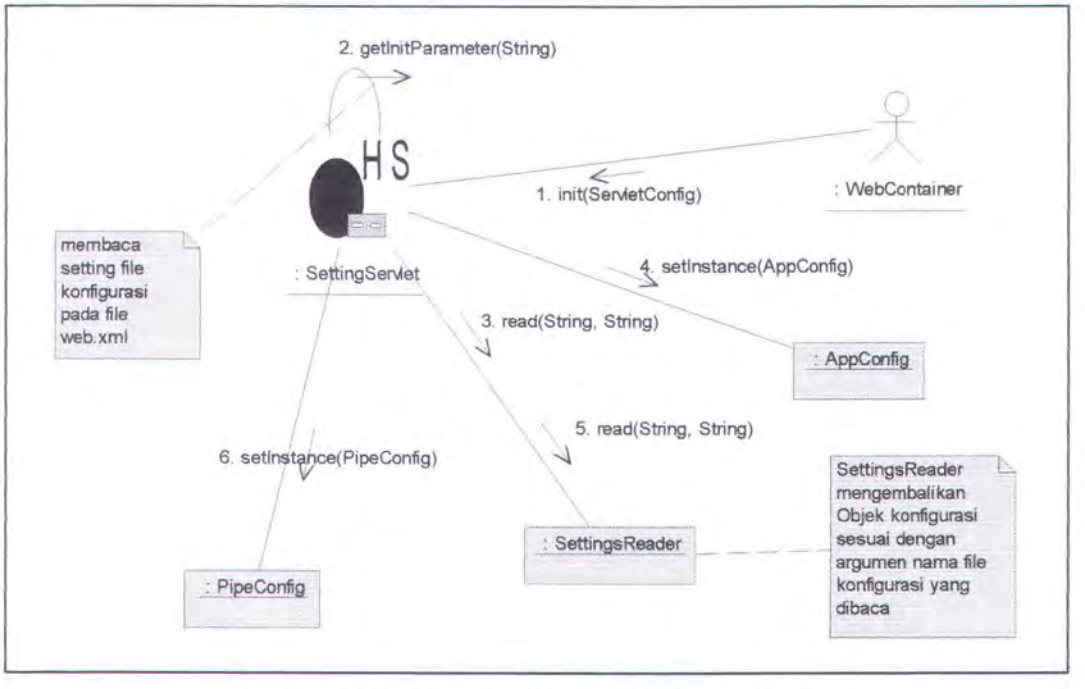
Gambar 3.13 *Class Diagram* keseluruhan aplikasi.

3.4.3. Collaboration Diagram

Untuk menggambarkan bagaimana kelas-kelas yang telah diinstansiasi berinteraksi pada saat *runtime*, perlu diberikan *Collaboration Diagram* dari aplikasi. Pada bagian ini diberikan diagram yang menggambarkan proses-proses penting dari aplikasi, proses-proses tersebut adalah: Application Startup, Request Processing, serta Bussiness Process.

3.4.3.1. Application Startup

Bagian ini memberikan gambaran proses yang berlangsung ketika aplikasi pertama kali dijalankan dan memuat konfigurasi aplikasi. Lihat gambar 3.14 di bawah.



Gambar 3.14 Collaboration Diagram Application Startup

Ketika pertama kali *web container* mulai dihidupkan, ia akan menginstansiasi SettingServlet dan memanggil fungsi `init()` dari servlet (1). Di dalam fungsi ini servlet mengambil nilai parameter-parameter dari file deskriptor

web.xml (2). Kemudian servlet membuat objek AppConfig untuk aplikasi menggunakan SettingsReader (3). Hasil dari pembacaan ini kemudian dijadikan sebagai argumen pemanggilan prosedur `setInstance()` dari AppConfig (4). Dengan pemanggilan prosedur ini, objek AppConfig siap dipakai oleh aplikasi.

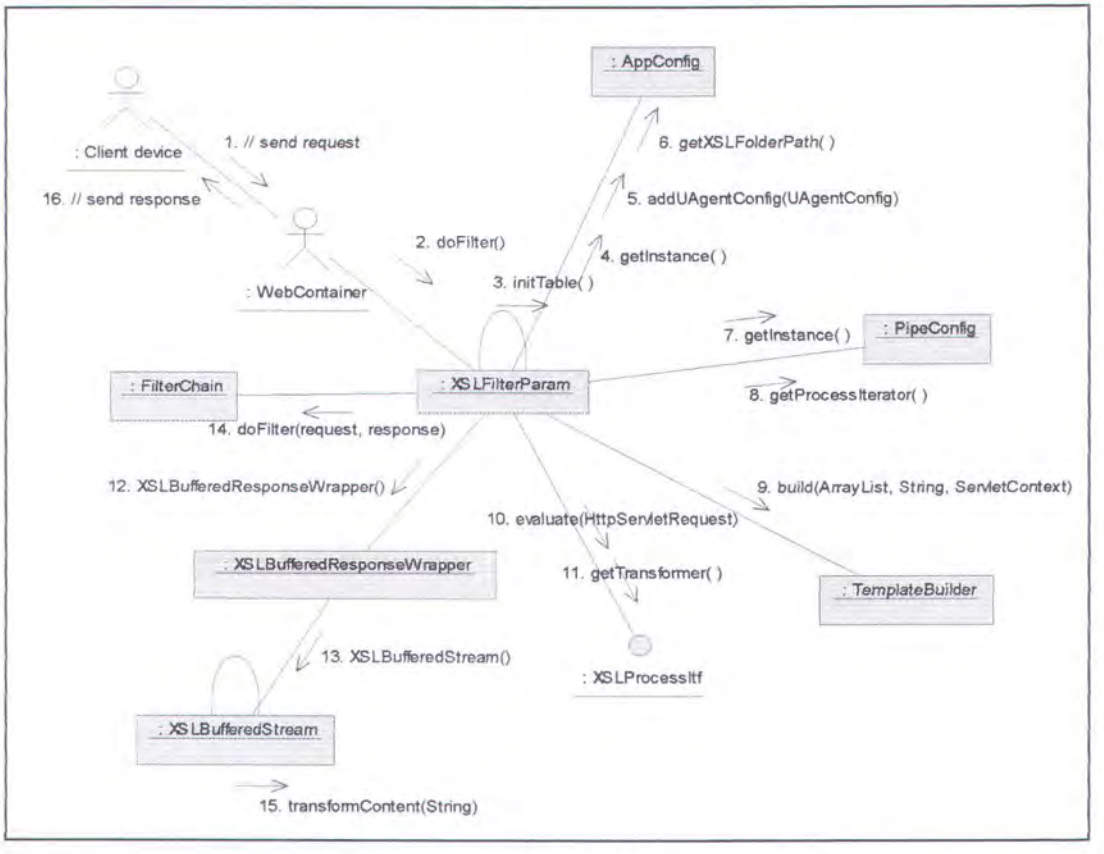
Selanjutnya servlet membaca file konfigurasi *pipeline* dengan SettingsReader (5). Hasil dari pembacaan ini adalah objek PipeConfig yang kemudian dijadikan sebagai argumen `setInstance()` PipeConfig (6). Di sini konfigurasi XML Pipeline siap digunakan.

3.4.3.2. Request Processing

Bagian ini menjelaskan bagaimana aplikasi mengolah *request* yang masuk dari *client*. Perbedaan utama antara arsitektur Model 2X dengan arsitektur aplikasi web konvensional yaitu adanya proses XML Pipeline yang mengolah luaran sebelum dikirim ke *client* dengan transformasi XML. Lihat gambar 3.15.

Proses ini diawali *client device* yang mengirim *request* ke *web container*. *Web Container* selanjutnya memanggil prosedur `doFilter()` dari servlet filter XSLFilterParam yang terpasang pada aplikasi. Untuk pemanggilan pertama kali, XSLFilterParam akan membangun tabel transformasi yang berisi daftar proses transformasi yang dilakukan sesuai dengan isi *request*. Ini dilakukan dengan pemanggilan prosedur `initTable()` (3). Di dalam prosedur ini XSLFilterParam akan menggunakan AppConfig untuk mendapatkan lokasi folder file XSL serta objek UAgentConfig yang membawa konfigurasi *device client* (4,5,6). Kemudian digunakan PipeConfig untuk mendapatkan konfigurasi XML Pipeline (7,8). Oleh

TemplateBuilder dari konfigurasi ini dibuat Transformer XML yang kemudian disimpan di dalam tabel transformasi (9).



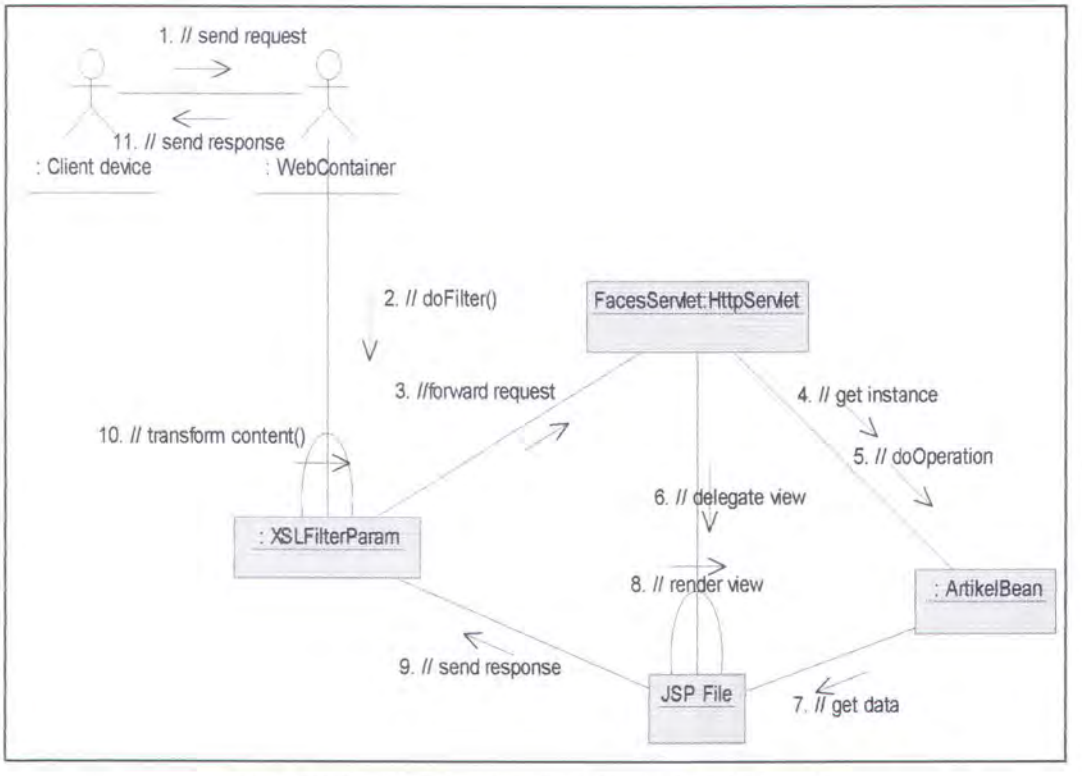
Gambar 3.15 Collaboration Diagram Request processing

Setelah tabel transformasi dibuat, diambil XML Transformer berdasarkan isi *request*. Ini dilakukan dengan prosedur `evaluate()` dan `getTransformer()` dari XSLProcessItf (10, 11). Selanjutnya ServletResponse yang diperoleh dibungkus dengan XSLBufferedResponseWrapper dan OuputStream servlet digantikan dengan XSLBufferedStream (12, 13). Kedua proses membungkus ini terjadi pada konstruktor XSLBufferedResponseWrapper. Proses ini diperlukan agar luaran tidak langsung dikirim ke browser, melainkan dapat diolah terlebih dahulu dengan transformasi XML.

Selanjutnya request dan response diteruskan ke FilterChain dengan method `doFilter()` (14). Hasil dari FilterChain adalah objek response yang akan dikirim ke *client*. Proses transformasi XML dilakukan terhadap objek response ini dengan pemanggilan prosedur `transformContent()` pada XSLBuffered Stream (15). Setelah ditransformasi, objek response ini lalu dikirim ke *client* (16).

3.4.3.3. Bussiness Process

Pada arsitektur Model2X, proses bisnis dilakukan oleh komponen Model MVC yang berperan sebagai data model. Untuk jelasnya, lihat gambar berikut.



Gambar 3.16 Collaboration Diagram Bussiness Process

Proses ini diawali oleh *client* yang mengirim *request* ke aplikasi, kemudian *web container* menerima *request* ini dan meneruskan ke aplikasi dengan pemanggilan fungsi doFilter() pada servlet filter XSLFilterParam (1, 2). Filter kemudian meneruskan *request* ke FacesServlet yang berperan sebagai *controller*

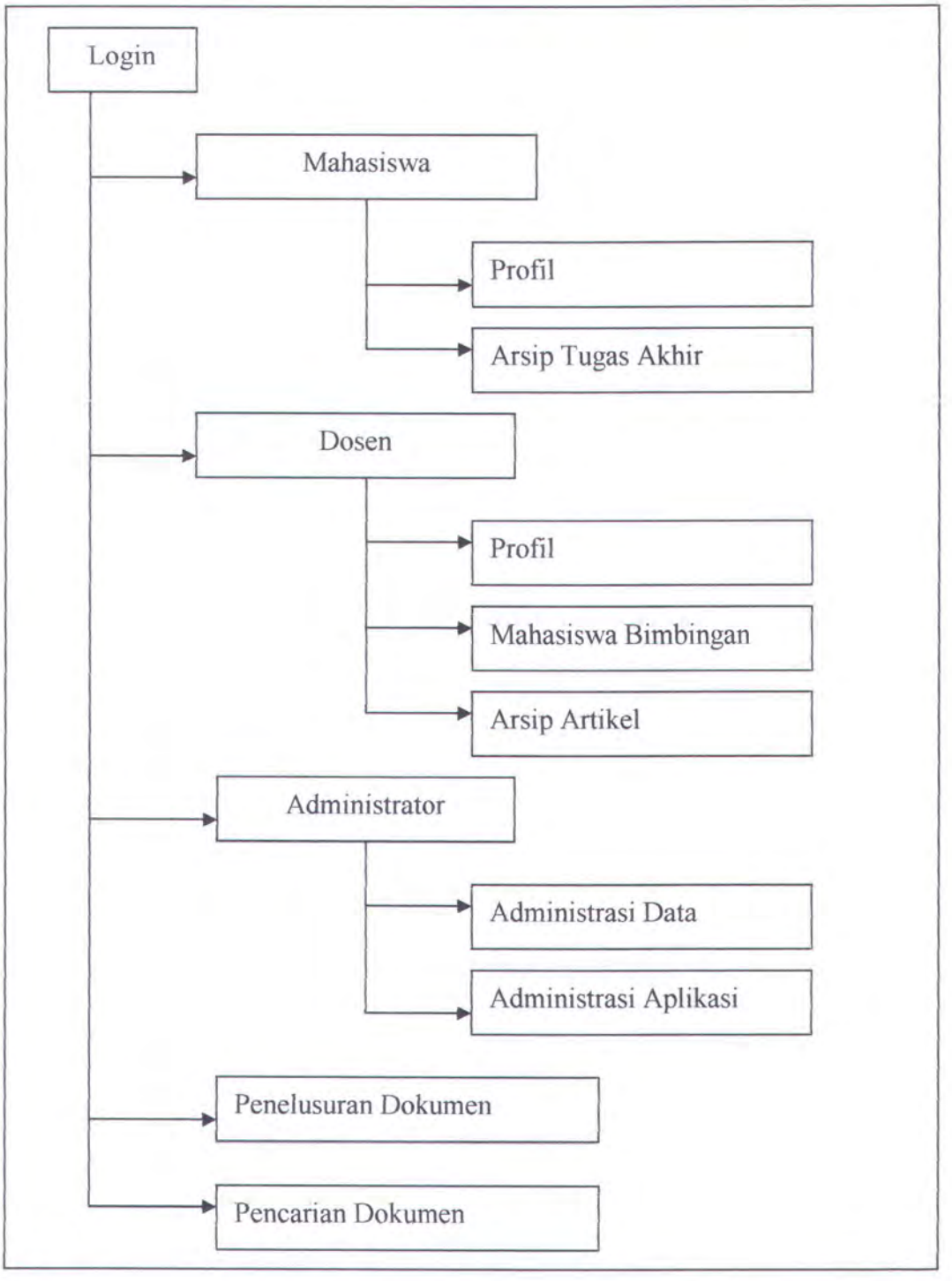
(3). FacesServlet menginstansiasi JavaBeans yang dibutuhkan serta mengeksekusi proses bisnis dari JavaBeans ini (4, 5). Sebagai contoh jika operasi yang dilakukan adalah memasukkan data ke database, maka setelah diinstansiasi JavaBeans akan menampung data-data dari form HTML yang akan dimasukkan ke database. Kemudian JavaBeans melakukan operasi *insert* data ke database dengan pemanggilan Action yang sesuai, yaitu getInsert.

Setelah proses bisnis dijalankan, FacesServlet kemudian meneruskan *request* bagian *view*, yaitu file JSP (6). JSP mengambil data-data yang dibutuhkan untuk ditampilkan ke layar jika ada (7). Selanjutnya JSP menuliskan halaman HTML sesuai proses yang telah berlangsung ke objek *response* (8). Objek response lalu dikirim ke XSLFilterParam (9). Sebelum dikirim kembali ke *client*, objek *response* akan diolah di XSLFilterParam sesuai dengan transformasi XML yang telah ditentukan (10). Proses pengolahan ini terjadi pada XSLBufferedStream. Selanjutnya hasil akhir dikirim ke *client* (11).

3.5. PERANCANGAN ANTARMUKA

Sub bab ini membahas mengenai perancangan antarmuka dari aplikasi web Pengarsipan Tugas Akhir. Menu-menu yang diperoleh pengguna tergantung dari jenis pengguna yang login ke sistem. Khusus untuk menu penelusuran, dapat diakses oleh semua pengguna. Diagram menu dapat dilihat pada gambar 3.17.

Untuk login mahasiswa, ada dua menu utama, yaitu Profil dan Arsip TA. Dari menu profil, mahasiswa dapat melihat data diri yang tercatat di database. Pada menu ini mahasiswa dapat memperbaharui data dirinya, dan juga melihat data dosen pembimbingnya. Pada menu Arsip TA



Gambar 3.17 Struktur Hirarki menu aplikasi

Untuk login dosen, ada tiga menu utama, yaitu profil, mahasiswa bimbingan, serta Artikel. Pada menu profil, dosen dapat melihat data diri, dan

merubah data diri. Menu mahasiswa bimbingan berisi daftar mahasiswa bimbingan Tugas Akhir dari dosen yang bersangkutan. Menu Artikel berisi daftar artikel yang telah di-*upload* ke database.

Untuk login Administrator ada dua kelompok menu utama, yaitu Administrasi data dan Administrasi aplikasi. Administrasi data terdiri dari: data mahasiswa, Arsip Tugas Akhir, Data Dosen, dan Arsip Artikel. Administrasi aplikasi terdiri dari Periode TA, Bidang Minat, dan Jenis Artikel.

BAB IV

IMPLEMENTASI PERANGKAT LUNAK

BAB IV

IMPLEMENTASI PERANGKAT LUNAK

Setelah tahap perancangan selesai, pada tahap implementasi perangkat lunak rancangan-rancangan yang telah dibuat diimplementasikan ke dalam bahasa pemrograman. Pada Tugas Akhir ini digunakan Bahasa Pemrograman Java.

4.1. IMPLEMENTASI DATA

Pembuatan tabel-tabel basidata dilakukan dengan perintah-perintah SQL yang termasuk dalam kategori *Data Definition Language* (DDL) yang ditulis dalam sebuah *SQL script* (Gambar 4.1).

Dengan demikian untuk pembuatan tabel dilakukan hanya dengan eksekusi *script* dengan menggunakan *tools* yang telah disediakan oleh Oracle seperti SQLPlus, atau SQL Worksheet. Tipe data yang digunakan dalam script pembuatan tabel merupakan tipe-tipe data dari database Oracle9i2.

```
create table TBIDANGMINAT  (
   BM_ID                INTEGER                         not null,
   BM_NAMA              VARCHAR2(50),
   BM_DESKRIPSI         LONG,
   constraint PK_TBIDANGMINAT primary key (BM_ID)
)
create table TDOSEN  (
   DSN_ID               INTEGER                         not null,
   DSN_NIP              NUMBER(10),
   DSN_NAMA             VARCHAR2(25),
   DSN_ALAMAT           VARCHAR2(25),
   DSN_TELP             VARCHAR2(20),
   DSN_LASTUPDATEDATE   DATE,
   DSN_PWD              VARCHAR2(10),
   constraint PK_TDOSEN primary key (DSN_ID)
)
create table TJENISARTIKEL  (
   JA_ID                INTEGER                         not null,
   JA_LABEL             VARCHAR2(25),
   JA_DESKRIPSI         VARCHAR2(100),
   constraint PK_TJENISARTIKEL primary key (JA_ID)) . . .
```

```
create table TPERIODETA  (
   PTA_ID               INTEGER                         not null,
   PTA_TANGGAL          DATE,
   PTA_JLH_MHS          INTEGER,
   constraint PK_TPERIODETA primary key (PTA_ID)
)
create table TARTIKEL  (
   ART_ID               INTEGER                         not null,
   JA_ID                INTEGER                         not null,
   DSN_ID               INTEGER                         not null,
   ART_JUDUL            VARCHAR2(50),
   ART_PENULIS          VARCHAR2(50),
   ART_DESKRIPSI        LONG,
   ART_KODE             VARCHAR2(25),
   ART_NAMA_FILE        VARCHAR2(25),
   ART_FILE             BLOB,
   ART_FLAG             VARCHAR2(25),
   ART_LASTUPDATEDATE   DATE,
   constraint PK_TARTIKEL primary key (ART_ID),
   constraint FK_TARTIKEL_MENGIRIM_TDOSEN foreign key (DSN_ID)
         references TDOSEN (DSN_ID),
   constraint FK_TARTIKEL_JENISARTI_TJENISAR foreign key (JA_ID)
         references TJENISARTIKEL (JA_ID)
)
create index MENGIRIM_FK on TARTIKEL (
   DSN_ID ASC
)
create index JENISARTIKEL_FK on TARTIKEL (
   JA_ID ASC
)
create table TMAHASISWA  (
   MHS_ID               INTEGER                         not null,
   PTA_ID               INTEGER                         not null,
   BM_ID                INTEGER                         not null,
   MHS_NRP              CHAR(10),
   MHS_NAMA             VARCHAR2(25),
   MHS_ALAMAT           VARCHAR2(50),
   MHS_ALAMAT_ASAL      VARCHAR2(50),
   MHS_AKT              CHAR(4),
   MHS_JUDUL_TA         VARCHAR2(50),
   MHS_ABSTRAK_TA       LONG,
   MHS_LASTUPDATEDATE   DATE,
   MHS_PWD              VARCHAR2(10),
   constraint PK_TMAHASISWA primary key (MHS_ID),
   constraint FK_TMAHASIS_BIDANGMIN_TBIDANGM foreign key (BM_ID)
         references TBIDANGMINAT (BM_ID),
   constraint FK_TMAHASIS_MASA_SIDA_TPERIODE foreign key (PTA_ID)
         references TPERIODETA (PTA_ID)
)
create index BIDANGMINATPILIHAN_FK on TMAHASISWA (
   BM_ID ASC
)
create index MASA_SIDANG_FK on TMAHASISWA (
   PTA_ID ASC
)
create table DIBIMBING  (
   MHS_ID               INTEGER                         not null,
   DSN_ID               INTEGER                         not null,
   constraint PK_DIBIMBING primary key (MHS_ID, DSN_ID),
   constraint FK_DIBIMBIN_DIBIMBING_TMAHASIS foreign key (MHS_ID)
         references TMAHASISWA (MHS_ID),
   constraint FK_DIBIMBIN_DIBIMBING_TDOSEN foreign key (DSN_ID)
         references TDOSEN (DSN_ID)
)
. . .
```

```
create index DIBIMBING_FK on DIBIMBING (
   MHS_ID ASC
)
create index DIBIMBING2_FK on DIBIMBING (
   DSN_ID ASC
)
/
create table TARSIPTA  (
   ATA_ID               INTEGER                          not null,
   MHS_ID               INTEGER,
   ATA_KETERANGAN       VARCHAR2(25),
   ATA_NAMA_FILE        VARCHAR2(25),
   ATA_FILE             BLOB,
   ATA_LASTUPDATEDATE   DATE,
   constraint PK_TARSIPTA primary key (ATA_ID),
   constraint FK_TARSIPTA_DATA_ARSI_TMAHASIS foreign key (MHS_ID)
         references TMAHASISWA (MHS_ID)
)
```

Gambar 4.1 Script SQL untuk pembuatan tabel.

Pembuatan Indeks Oracle Text

Pada Aplikasi ini Tabel yang menyimpan dokumen adalah tabel TArsipTA dan tabel TArtikel. Tabel ini berperan sebagai *Text Table.* Untuk memanfaatkan fasilitas Oracle Text perlu dibuat sebuah indeks Oracle Text pada masing-masing tabel tersebut, yaitu pada kolom yang menyimpan dokumen. Indeks Oracle Text dipasang pada kolom TArsipTA.Ata_file dan kolom TArtikel.Art_file.

Pada gambar 4.2. berikut diberikan perintah SQL yang digunakan untuk membuat indeks pada kedua tabel di atas. Jenis filter yang dipakai adalah filter INSO_FILTER. Untuk membuat *preference* filter ini digunakan perintah SQL pada gambar 4.2.

```
create index arsip_text_idx on TArsipTA ( ata_file )
  indextype is ctxsys.context
  parameters ('FILTER INSO_FILTER_PREF');

create index art_filter_idx on TArtikel ( art_file )
  indextype is ctxsys.context
  parameters ('FILTER INSO_FILTER_PREF');
```

Gambar 4.2 Membuat Indeks Oracle Text

```
begin
  Ctx_Ddl.Create_Preference
    (
      preference_name => 'INSO_FILTER_PREF',
      object_name     => 'INSO_FILTER'
    );
end;
/
```

Gambar 4.3 Membuat *preference* filter

4.2. IMPLEMENTASI PROSES

Pada bab ini dijelaskan beberapa kelas-kelas penting yang berperan dalam proses inti dari aplikasi. Penjelasan kelas-kelas dikelompokkan berdasarkan *Package* yang menampung kelas tersebut, sebagaimana diperlihatkan pada *Package Diagram* gambar 3.9.

4.2.1. Package formbean

Kelas-kelas yang terdapat pada Package formbean adalah kelas-kelas yang berfungsi sebagai data model. Pada Framework JSF, kelas-kelas ini dibutuhkan setiap kali membuat *form* yang memerlukan operasi-operasi *logic*. Contohnya operasi menambah atau menghapus data dari database.

4.2.1.1. MahasiswaBean

Kelas ini memodelkan mahasiswa yang mengakses aplikasi ini. Method-method penting kelas ini dapat dilihat pada gambar 4.4. Atribut dari kelas ini disesuaikan dengan field tabel TMahasiswa. Semua atribut kelas ini memiliki setter dan getter untuk mengubah nilai atributnya. Contoh method getter adalah `getMhs_id()` dan `getMhs_nama()`. Sedangkan setter misalnya `setMhs_id()` dan `setMhs_nama()`. Untuk operasi insert dan update data di database, kelas ini memiliki method yang memiliki nilai kembali kelas

`javax.faces.application.Action`. Objek yang dihasilkan method ini berisi proses bisnis yang akan dijalankan oleh JSF.

```
void   deleteData(java.lang.String p_id)
Action getInsert()
Action getUpdate()
ArrayList    init_data(java.lang.String where_clause)
void   load_data(java.lang.String p_id)
void   resetData()
...
 String getMhs_id()
 String getMhs_judul_ta()
 Timestamp getMhs_lastupdatedate()
 String getMhs_nama()
 String getMhs_nrp()
 String getMhs_pwd()
...
 void setMhs_id(java.lang.String i)
 void setMhs_judul_ta(java.lang.String string)
 void setMhs_lastupdatedate(java.sql.Timestamp date)
 void setMhs_nama(java.lang.String string)
 void setMhs_nrp(java.lang.String string)
 void setMhs_pwd(java.lang.String string)
...
```

Gambar 4.4 Method-method pada kelas TMahasiswa.

Method `deleteData()`, `getInsert()`, dan `getUpdate()` berfungsi untuk operasi database. Method `init_data()` dan `load_data()` berfungsi untuk mengambil data dari database. Method `resetData()` berfungsi mengosongkan atribut kelas ini. Sisanya adalah setter dan getter yang berfungsi untuk mengubah nilai atribut kelas ini

Kelas DosenBean memiliki struktur yang serupa dengan kelas MahasiswaBean, yaitu memiliki atribut yang bersesuaian dengan tabel TDosen, memiliki setter dan getter untuk atribut-atribut tersebut, memiliki method `deleteData()`, `getInsert()`, `getUpdate()`, `resetData()` `init_data()` dan `load_data()` yang berfungsi untuk operasi database

4.2.1.2. Kelas ArsipTABean

Kelas ini memodelkan Arsip Tugas Akhir mahasiswa, bersesuaian dengan tabel TArsipTA. Field-field pada tabel TArsipTA memiliki atribut yang bersesuaian pada kelas ini. Berbeda dengan kelas MahasiswaBean dan DosenBean, kelas ini tidak digunakan secara langsung oleh JSF, melainkan digunakan sebagai JavaBeans pada file JSP.

Gambar 4.5 berikut memperlihatkan method-method yang dimiliki oleh kelas ini.


```
void deleteData(java.lang.String p_ata_id)

String getATA_id()

String getATA_Keterangan()
...
java.util.ArrayList init_data(String where_clause)

void load_data(String p_id)

void setATA_id(String i)

void setATA_Keterangan(String string)
...
void updateData(String p_nextid, String p_ktg, String p_mhs_id)
```

Gambar 4.5 Method-method pada kelas ArsipTABean

Kelas ArtikelBean memiliki fungsi yang serupa dengan kelas ArsipTABean, yaitu memodelkan data Artikel yang di-*upload* ke database.

Method-method yang dimiliki oleh kelas ini hampir sama dengan method yang dimiliki kelas ArsipTABean. Untuk akses ke database yaitu `updateData()`, `load_data()`, `init_data()`, dan `deleteData()`. Selebihnya adalah method setter dan getter untuk mengubah nilai atribut.

4.2.2. Package data

Kelas-kelas di *package* ini berfungsi sebagai enkapsulasi operasi-operasi ke database. Dengan Kelas-kelas ini, operasi database lebih mudah dilakukan, baik untuk mengambil data, menambah, merubah, ataupun menghapus data. Ada dua kelas di *package* ini, yaitu DBConnection, dan DataSource.

Kelas DBConnection berperan untuk menyimpan koneksi ke database dan kelas DataSource berperan untuk melakukan operasi-operasi database. Pada gambar 4.6 berikut diberikan method-method pada kelas DBConnection. Untuk kelas DataSource diberikan pada gambar 4.7.

```
java.sql.Connection getConnection()
static DBConnection getInstance()
void setConfiguration(DatabaseConfig dbc)
void setConnURL(java.lang.String url)
void setDbdriver(java.lang.String dbdriver)
void setDbpassword(java.lang.String dbpassword)
void setDbusername(java.lang.String dbusername)
```

Gambar 4.6 Method pada kelas DBConnection

Untuk menggunakan kelas ini, terlebih dahulu diambil objek DBConnection dengan method `getInstance()`, lalu koneksi database diambil dengan method `getConnection()`.

```
void clearData()
 void commit()
 void disconnect()
 int executeQuery(java.lang.String sql)
 void executeUpdate(java.lang.String sql)
 java.util.Vector getColumnHeads()
 int getNumCols()
 int getNumRows()
 java.util.Vector getRows()
 java.util.Vector getSingleRow()
 java.lang.Object getSingleVal()
 void setup()
```

Gambar 4.7 Method pada kelas DataSource

Kelas DataSource ini menggunakan kelas DBConnection untuk mengambil koneksi ke database. Untuk menggunakan kelas ini terlebih dahulu dipanggil method `setup()`, kemudian mengeksekusi perintah SQL melalui method `execute()` dan `executeQuery()`. Hasil eksekusi berupa objek Vector, diambil dengan salah satu method: `getRows()`, `getSingleRow`, atau `getSingleVal()`.

4.2.3. Package filter

Package ini berisi kelas-kelas yang berfungsi untuk menyusun proses XML Pipeline. Pada bagian ini akan dijelaskan dua kelas yang paling penting, yaitu XSLBufferedStream dan XSLFilterParam.

4.2.3.1. XSLFilterParam

Kelas ini berfungsi sebagai ServletFilter tempat pengolahan data dari web container sebelum dikirim ke *client*. Pada gambar berikut diberikan potongan kode method utama, yaitu `doFilter()` dari kelas ini.

```
public class XSLFilterParam implements Filter {
  public void doFilter(ServletRequest request,
                                    ServletResponse response,
                                    FilterChain chain)
                      throws IOException, ServletException {
  ...
            for (Iterator iter = this.processList.iterator();
                  iter.hasNext();)
                  {
                  Transformer xsltTransformer = null;

                  XSLProcessItf process =
                                          (XSLProcessItf) iter.next();
                  process.evaluate((HttpServletRequest)request);
                  xsltTransformer = process.getTransformer();

                  if(xsltTransformer!=null)
                  {
                        empty_flag = false;
                        myList.add(xsltTransformer);
                  }
            }// end for

            if(!empty_flag){
                  XSLBufferedResponseWrapper myWrappedResp = new
XSLBufferedResponseWrapper( response, myList);
                  chain.doFilter(request,  myWrappedResp);
                  myWrappedResp.getOutputStream().close();
            }
            else
            {
            chain.doFilter(request, response);
            }
  }
  ...
```

Gambar 4.8 Potongan Kode Program Kelas XSLFilterParam

Dari gambar di atas, khusus pada bagian yang tercetak tebal, diperlihatkan bagaimana Kelas XSLFilterParam mengumpulkan transformer sebagai satuan proses transformasi ke dalam sebuah list. Sejumlah transformer inilah yang nantinya digunakan oleh *wrapper* OutputStream dalam proses transformasinya.

Untuk setiap XSLProcessItf didalam processList dihasilkan sebuah transformer. Pemilihan transformer ini berdasarkan hasil pemanggilan method `XSLProcessItf.evaluate()`. Method ini membaca isi variabel session dari request. variabel yang dibaca sesuai dengan parameter yang menjadi acuan proses ini. Misalkan untuk proses yang mengolah penggunaan bahasa, didefiniskan variabel *lang* sebagai parameter untuk proses ini. Maka isi variabel session dengan nama *lang* akan menentukan transformer yang dihasilkan oleh kelas XSLProcessItf.

Setelah diperoleh sebuah ArrayList transformer, objek response lalu dibungkus dengan Kelas XSLBufferedResponseWrapper. Nantinya kelas inilah yang dikirim ke servlet dalam *web container*.

Pemanggilan method `close()` pada objek *wrapper* response bertujuan untuk mengeksekusi. proses transformasi XML.

4.2.3.2. XSLBufferedStream

Implementasi proses transformasi XML di *package* ini terdapat pada kelas *wrapper* ServletResponse dan *wrapper* ServletOutputStream yaitu XSLBufferedResponseWrapper dan XSLBufferedStream. Kedua kelas ini terletak pada *package* `filter.wrap`. Pada gambar berikut diberikan potongan kode implementasi transformasi XML pada method `transformContent()` di kelas XSLBufferedStream.

Pada method ini proses transformasi berada di dalam sebuah loop yang mengambil isi ArrayList transList. Proses pengolahan data oleh objek trans dapat berlangsung beberapa kali, tergantung jumlah transformer di objek transList. Transformer ini diperoleh dari kelas XSLFilterParam.

```
public String  transformContent(String inStr) {

  String retVal =""; String firstPart="";
  StringWriter swPipeOutput = null;
  String strPipeInput = new String(inStr);

  Iterator iter = this.transList.iterator();

  while (iter.hasNext()) {
       Transformer trans = (Transformer) iter.next();
       swPipeOutput = new StringWriter();
       try {

       trans.transform(new StreamSource( new
StringReader(strPipeInput)), new StreamResult(swPipeOutput));

       } catch (TransformerException e) {
            System.out.println(e.getMessageAndLocation());
       }
       strPipeInput = new String(swPipeOutput.getBuffer());
  } // end while

  retVal = new String(swPipeOutput.getBuffer());
  return retVal;

}
```

Gambar 4.9. Method transformContent() pada kelas XSLBufferedStream

4.2.4. Package settings

Package ini berisi kelas-kelas yang bertugas menyediakan informasi setting aplikasi bagi kelas-kelas lain secara keseluruhan. Untuk konfigurasi aplikasi, digunakan format XML dengan tujuan memudahkan pembacaan. Ada dua file yang digunakan sebagai tempat konfigurasi aplikasi, yaitu file `pta-app-config.xml` dan `pta-pipe-config.xml`. File yang pertama berfungsi sebagai tempat konfigurasi aplikasi secara umum. File kedua berfungsi untuk konfigurasi proses transformasi. Gambar 4.10 dan 4.11 adalah potongan dari kedua file konfigurasi.

Pada sub bab ini akan dibahas tiga kelas yang paling berperan dalam proses penyediaan setting aplikasi ini, yaitu SettingsServlet, SettingsReader, dan kelas AppConfig.

```
<application-settings>
 <app-name> FPR </app-name>
 <app-desc> Final Project Repository </app-desc>
 <database>
       <name>MyDB-1</name>
       <driver>oracle.jdbc.OracleDriver</driver>
       <url>jdbc:oracle:thin:@127.0.0.1:1521:PRIMADB</url>
       <user>pta</user>
       <password>ptapwd</password>
 </database>
 . . .
```

Gambar 4.10 Potongan file konfigurasi pta-app-config.xml

```
<pipe-config>
       <pipe-name>pipe01</pipe-name>
<process-param>
       <name>theme-config</name>
       <param-name>theme</param-name>
       <param-value>common-html</param-value>
       <param-value>simple-html</param-value>
</process-param>
  . . .
```

Gambar 4.11 Potongan file konfigurasi pta-pipe-config.xml

4.2.4.1. SettingServlet

Kelas ini berupa Servlet yang menginisialisasi proses pembacaan konfigurasi aplikasi. Agar Servlet ini diaktifkan diawal berjalannya aplikasi, maka pada file deskriptor aplikasi ini, yaitu file `web.xml`, SettingServlet diberi nilai 2 untuk elemen load-on-startup. Lihat gambar 4.12 untuk potongan file deskriptor web.xml. Artinya SettingServlet mendapat prioritas kedua untuk diaktifkan ketika aplikasi pertama kali dijalankan. Prioritas pertama diberikan kepada FacesServlet yang merupakan salah satu komponen dari JSF.

```
    . . .
  <servlet>
      <servlet-name>Setting-Servlet</servlet-name>
      <servlet-class>settings.SettingServlet</servlet-class>
      <load-on-startup>2</load-on-startup>
  </servlet>
    . . .
```

Gambar 4.12 Nilai setting servlet pada file web.xml

Lihat gambar 4.13. Proses pembacaan konfigurasi dimulai dengan membaca nilai parameter inisial *servlet context*. Nilai parameter yang diambil adalah lokasi dari file konfigurasi yang akan dibaca. Lokasi file konfigurasi ini kemudian dijadikan argumen dari Kelas SettingsReader utnuk menghasilkan objek konfigurasi yang dapat dipakai. Untuk mengisi kelas AppConfig dan kelas PipeConfig dengan objek konfigurasi yang sudah jadi digunakan method `setInstance()` pada kedua kelas ini.

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

public class SettingServlet extends HttpServlet {

 public void init(ServletConfig p_conf) throws ServletException {
       . . .
       rootFile = ctx.getInitParameter ("AppConfigFile");
       confpath = ctx.getRealPath(rootFile);
       rootFile = ctx.getInitParameter ("AppRuleFile");
       rulepath = ctx.getRealPath(rootFile);
       xslpath = ctx.getInitParameter ("xslPath");
       AppConfig.setInstance( (AppConfig)
             SettingsReader.read(confpath, rulepath) );
       AppConfig.getInstance().setXSLFolderPath(xslpath);
       rootFile = ctx.getInitParameter ("PipeConfigFile");
       confpath = ctx.getRealPath(rootFile);
       rootFile = ctx.getInitParameter ("PipeRuleFile");
       rulepath = ctx.getRealPath(rootFile);
       PipeConfig.setInstance( (PipeConfig)
             SettingsReader.read(pipeconfpath, piperulepath) );
       . . .
       }
}
```

Gambar 4.13 Potongan Kode SettingServlet pada method init().

4.2.4.2. SettingsReader

Kelas ini berfungsi untuk membaca file konfigurasi yang lokasi filenya diberikan oleh SettingServlet. Kelas ini hanya mempunyai sebuah method statik yang berfungsi untuk proses pembacaan. Lihat gambar 4.14. Pada kode program

yang tercetak tebal, dilakukan proses pembacaan file konfigurasi. Pembacaan file ini menggunakan kelas Digester yang merupakan salah satu *library* dari Apache Commons. Kelas Digester menggunakan method `Digester.parse()` untuk mengolah file xml menjadi objek yang berisikan data-data pada file xml tersebut. Proses pembacaan file ini membutuhkan sebuah file xml *digester-rule.* File ini berisi petunjuk bagi Digester tentang Kelas yang harus diinstansiasi, serta method yang dipakai untuk memuat data kedalam objek hasil. Gambar 4.15 menunjukkan potongan file xml *digester-rule.* Gambar 4.16 menunjukkan potongan file konfigurasi yang dibaca.

```
/* [fprApp : SettinsReader.java]
 * Created on Jun 22, 2004 : 4:42:18 PM
 *
 */
package settings;

import java.io.File;

import org.apache.commons.digester.Digester;

import org.apache.commons.digester.xmlrules.DigesterLoader;

public class SettingsReader {
      public static Object read
            (String configFile, String ruleFile)
      {Object retObject = null;
        try{
              File inputXMLFile = new File(configFile);
              File rules = new File(ruleFile);

              Digester digester =
                  DigesterLoader.createDigester(rules.toURL());

            retObject = digester.parse(inputXMLFile);

        }catch(Exception e){
                  e.printStackTrace();
        }
        return retObject;
      }
}
```

Gambar 4.14 Kode program Kelas SettingsReader

```
<digester-rules>
   <object-create-rule pattern="application-settings"
                                  classname="settings.AppConfig" />
  <call-method-rule pattern="application-settings/app-name"
                    methodname="setAppName" paramcount="0" />
   <call-method-rule pattern="application-settings/app-desc"
                   methodname="setAppDesc" paramcount="0" />

   <pattern value="application-settings/database">
      <object-create-rule
             classname="settings.settingmodel.DatabaseConfig" />
      <call-method-rule pattern="name" methodname="setName"
                        paramcount="0" />
      <call-method-rule pattern="driver" methodname="setDriver"
                        paramcount="0" />
      <call-method-rule pattern="url" methodname="setUrl"
                        paramcount="0" />
      <call-method-rule pattern="user" methodname="setUser"
                        paramcount="0" />
      <call-method-rule pattern="password" methodname="setPassword"
                        paramcount="0" />
      <set-next-rule methodname="addDBConfig" />
   </pattern>
. . .
```

Gambar 4.15 Potongan file xml digester-rule untuk kelas AppConfig

```
<application-settings>
 <app-name> FPR </app-name>
 <app-desc> Final Project Repository </app-desc>
 <database>
       <name>MyDB-1</name>
       <driver>oracle.jdbc.OracleDriver</driver>
       <url>jdbc:oracle:thin:@127.0.0.1:1521:PRIMADB</url>
       <user>pta</user>
       <password>ptapwd</password>
 </database>
 <database>
       <name>MyDB-1</name>
       <driver>oracle.jdbc.OracleDriver</driver>
       <url>jdbc:oracle:oci8:@PRIMADB</url>
       <user>pta</user>
       <password>ptapwd</password>
 </database>
 . . .
```

Gambar 4.16 Potongan file konfigurasi pta-app-config.xml

4.2.4.3. AppConfig

Kelas ini berisi representasi konfigurasi aplikasi secara keseluruhan. Pembuatan objek dari kelas ini dilakukan di SettingServlet oleh SettingsReader.

Untuk menggunakan kelas ini, terlebih dahulu diambil instansiasi kelas ini dengan menggunakan method `getInstance()`. Untuk mempermudah akses ke kelas ini, digunakan pattern singelton, di mana untuk seluruh aplikasi hanya terdapat satu buah objek AppConfig, dan objek ini dapat diperoleh dengan sebuah method statik getInstance(). Berikut ini potongan kode program Kelas AppConfig.

```
. . .
public class AppConfig {

 private static boolean isSetup = false;
 private static AppConfig m_Instance;
 private String appName;
 private String appDesc;

 private String XSLFolderPath;
 //list of DatabaseConfig;
 private ArrayList DBConfig = new ArrayList();
 . . .
public static void setInstance(AppConfig app)
 {
       if(isSetup == false)
       {
             isSetup = true;
             System.out.println("instance setup!");
             m_Instance = app;
       }else
       {
             System.out.println("cannot setup instance twice !");
       }
 }
 public static AppConfig getInstance()
 {
       if(m_Instance == null)
       {
             System.out.println(" failed !! instance has not been
set up");
       }
       return m_Instance;
 }

public String getAppDesc() {
       return appDesc;
}

public String getAppName() {
       return appName;
}
 . . .
```

Gambar 4.17. Potongan kode program kelas AppConfig

Pembuatan XMLRenderKit

RenderKit adalah sekumpulan kelas java yang berfungsi untuk menampilkan bentuk objek Antarmuka. RenderKit yang disediakan oleh Sun Microsystem dalam paket Java Web Service Developer Kit adalah RenderKit untuk menampilkan objek UI dalam bahasa HTML.

Pada Arsitektur Model 2X dibutuhkan format keluaran dari file-file jsp yang berbentuk XML, atau XML *compatible*. Hal ini dimaksudkan agar keluaran dsri file jsp dapat diolah dengan transformer XML.

Pembuatan XMLRenderKit dapat dilakukan dengan mengganti bagian-bagian dari RenderKit HTML ke dalam format yang sesuai dengan spasifikasi XML. Sebagai perbandingan, berikut diberikan potongan kode RenderKit HTML (Gambar 4.18) dan RenderKit XML (Gambar 4.19). Kelas Render ini menggambar sebuah objek input hidden.

Pada potongan kode yang bercetak tebal terlihat bahwa karakter penutup pada RenderKit HTML adalah "**>**". Agar sesuai dengan spesifikasi XML, maka karakter penutup ini diganti dengan string "**/>**".

```
. . .
 buffer.append("<input type=\"hidden\"");
 buffer.append(" name=\"");
 buffer.append(component.getClientId(context));
 buffer.append("\"");
 if(currentValue != null)
 {
     buffer.append(" value=\"");
     buffer.append(currentValue);
     buffer.append("\"");
 }
 buffer.append(">");
. . .
```

Ganbar 4.18 RenderKit input hidden.

```
. . .
buffer.append("<input type=\"hidden\"");
buffer.append(" name=\"");
buffer.append(component.getClientId(context));
buffer.append("\"");
if(currentValue != null)
{
    buffer.append(" value=\"");
    buffer.append(currentValue);
    buffer.append("\"");
}
buffer.append("/>");
. . .
```

Gambar 4.19 RenderKit input hidden disesuaikan dengan format XML

4.3. IMPLEMENTASI ANTARMUKA

Pada Model 2X, pembuatan antarmuka dibedakan menjadi dua bagian, yaitu pembuatan template berupa file XSL, serta pembuatan halaman web dalam file JSP. Dalam tugas akhir ini digunakan teknologi JSF untuk membantu pembuatan Antarmuka. Keluaran dari JSP adalah dalam bentuk XML yang tidak memiliki format tampilan. Format tampilan ditangani oleh template XSL.

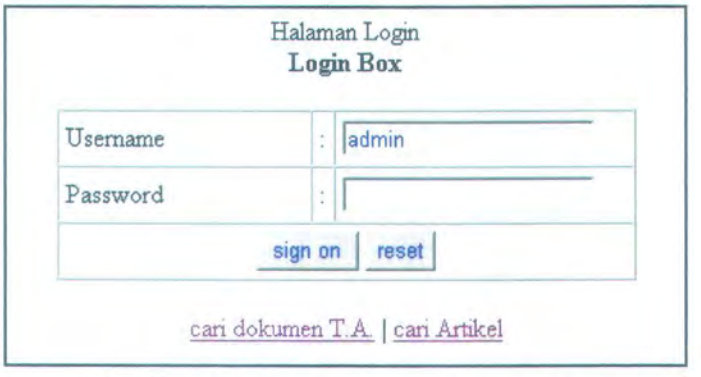
Berikut ini potongan template XSL yang digunakan beserta tampilannya.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
<xsl:template match="/">
...
            <div align="left">
                  <!-- judul -->
                  Pengarsipan Tugas Akhir
                    </div>
...
        <tr>
          <td valign="top">
         <xsl:value-of select="root/header"/><hr/>
                  <xsl:copy-of select="root/content"/>
                  </td>
        </tr>
...
            </body>
            </html>
</xsl:template>
</xsl:stylesheet>
```

Gambar 4.20 Potongan template XSL

Gambar 4.21 Tampilan template XSL

Isi dari file JSP adalah isi halaman web tanpa format tampilan. Berikut ini contoh luaran JSP tanpa format tampilan



Gambar 4.22 Tampilan JSP tanpa template XSL

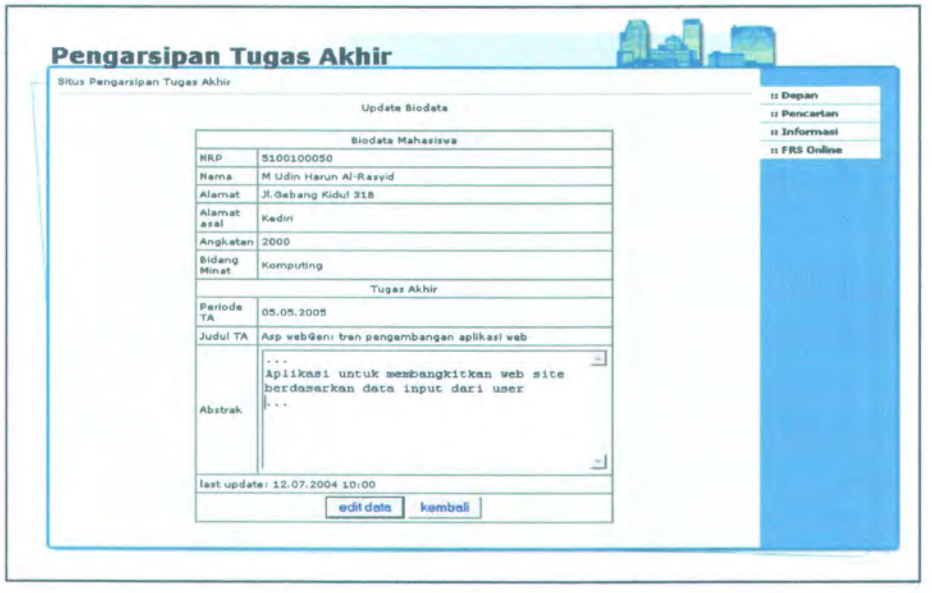
Berikut ini beberapa tampilan antarmuka implementasi rancangan bab sebelumnya.

Gambar 4.23 Tampilan untuk daftar Arsip Tugas Akhir

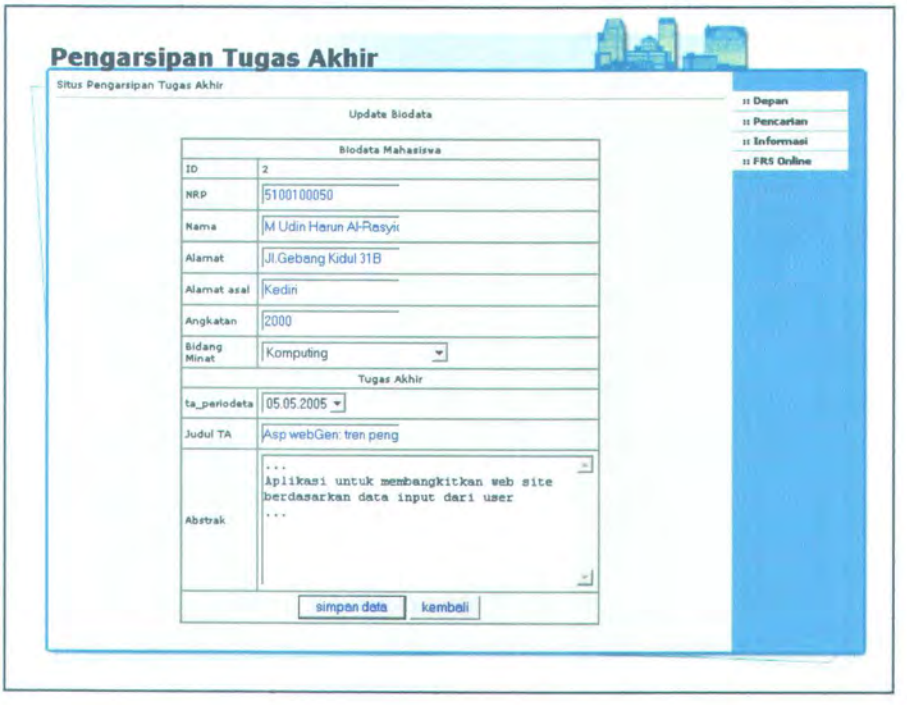
```
<%@ taglib uri="http://java.sun.com/jsf/core"    prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html"    prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/demo/components" prefix="d"
%>
<header>
      <center>::Situs Pengarsipan Tugas Akhir::</center>
</header>

<content>
<f:use_faces>
<d:panel_resultset columnClasses="list-column-01,list-column-
03,list-column-04,list-column-05"
                  headerClass="list-header"
                   panelClass="list-background"
                   rowClasses="list-row-even,list-row-odd"
                   navFacetOrientation="WEST"
                   rowsPerPage="5">

                   <f:facet name="header">
                  <h:panel_group>
                    <h:output_text value="Nama file"/>
                    <h:output_text value="Last update date"/>
                    <h:output_text value="[download]"/>
                       <h:output_text value="[delete]"/>

                  </h:panel_group>
                   </f:facet>
...
             <h:panel_data var="ab"
                       valueRef="list_arsip_ta">
             <h:output_text id="namafile1"
                       valueRef="ab.ATA_Namafile"/>
             <h:output_text id="updatedate1"
                       valueRef="ab.strLastupdatedate"/>
              <h:command_hyperlink id="hrefParamLink2"
      commandName="mylink2" href="adm_mhs_filelist_download.jsp"
      label="[download]">
          <f:parameter id="Param2" name="p_id"
                          valueRef="ab.ATA_id"/>
      </h:command_hyperlink>
      <h:command_hyperlink id="hrefParamLink3"
      commandName="mylink3" href="adm_mhs_filelist_delete.jsp"
                    label="[delete]">
    <f:parameter id="Param3" name="p_ata_id" valueRef="ab.ATA_id"/>
      </h:command_hyperlink>
    </h:panel_data>
. . .

</f:use_faces>
</content>

</root>
```

Gambar 4.24 Kode JSP untuk daftar Arsip Tugas Akhir

Berikut ini beberapa tampilan dari aplikasi setelah antarmuka aplikasi diimplementasikan.

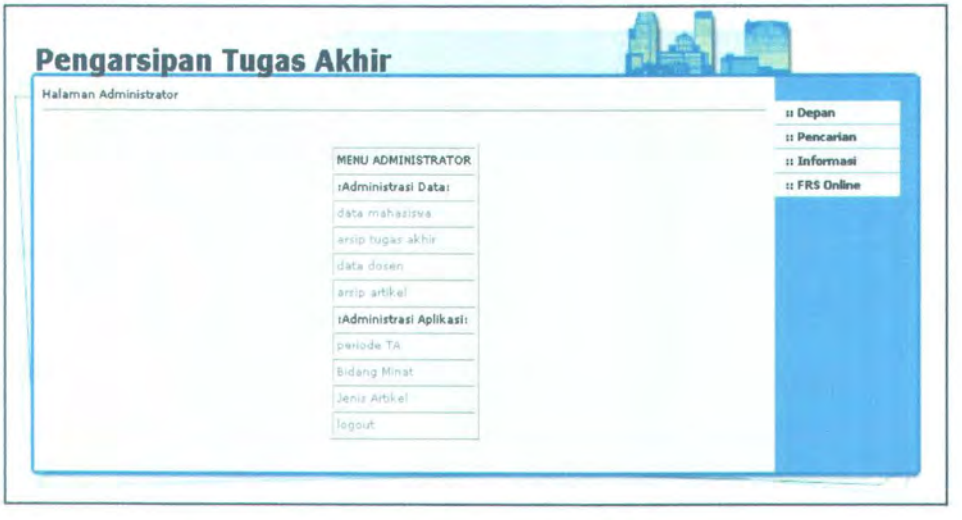


Gambar 4.25 Tampilan melihat data mahasiswa



Gambar 4.26 Tampilan merubah data mahasiswa

Gambar 4.27. Tampilan melihat arsip Tugas Akhir.



Gambar 4.28. Tampilan menu utama Administrator

BAB V

UJI COBA DAN EVALUASI

- Client

Sedangkan untuk *client*, tidak ada spesifikasi khusus. Yang dibutuhkan adalah komputer yang mendukung memiliki *browser* Internet seperti Internet Explorer atau yang lainnya.

5.2. SKENARIO UJI COBA

Pada sub bab ini diberikan beberapa skenario untuk menjalankan uji coba guna menguji keberhasilan implementasi perangkat lunak.

5.2.1. Skenario Pertama

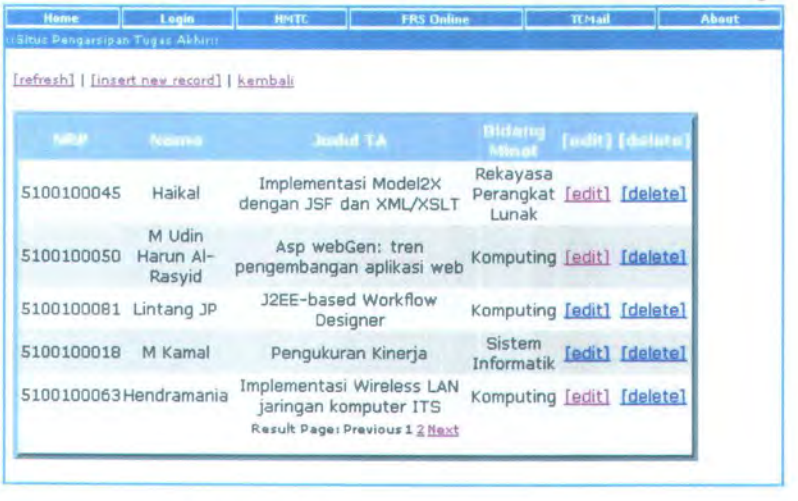
Skenario pertama digunakan untuk menguji apakah arsitektur model 2X yang telah diterapkan berjalan dengan baik. Pada skenario ini akan dilakukan proses login ke dalam aplikasi, dilanjutkan dengan melihat data pengguna dan mengubah data pengguna.

Gambar 5.1 berikut ini tampilan layar login dengan theme "blueTheme" pada aplikasi pengarsipan tugas akhir. Setelah malakukan proses login sebagai administrator, ditampilkan menu-menu administrator, seperti pada gambar 5.2.

Gambar 5.1 Layar login

Gambar 5.2 Layar menu utama administrator

Pada menu administrator terdapat menu-menu yang berhubungan dengan data aplikasi (Administrasi Data), dan menu-menu yang berhubungan dengan nilai *setting* aplikasi (Administrasi Aplikasi). Kemudian diklik menu data mahasiswa.

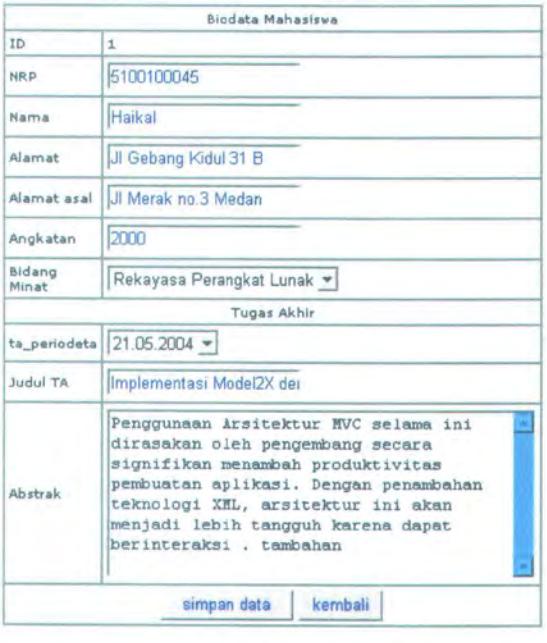


Gambar 5.3 Tampilan daftar mahasiswa

Dari daftar ini kemudian dipilih salah satu nama mahasiwa. Aplikasi akan menampilkan data detail mahasiswa.

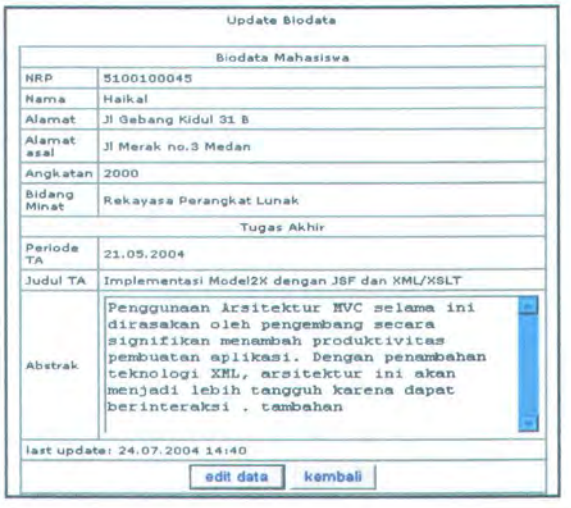
Gambar 5.4 Tampilan data detail mahasiswa

Untuk mengubah data detail, diklik tombol "edit data". Aplikasi akan menampilkan form update data.



Gambar 5.5. Form merubah data

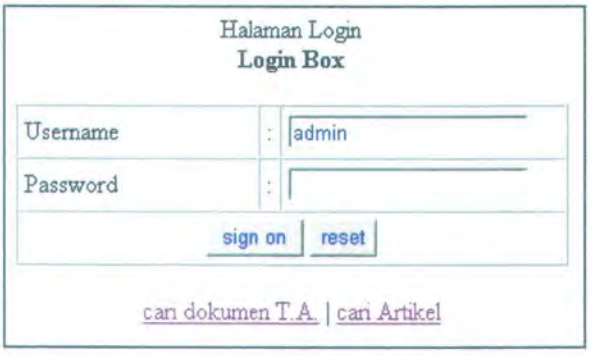
Setelah merubah data, diklik tombol "simpan data". Aplikasi menampilkan data terbaru.



Gambar 5.6 Hasil perubahan data

Proses update data ini menggunakan javabean dalam melakukan proses-proses operasi database, baik proses insert, update, maupun delete. Di mana javabeans ini dijadikan sebagai *formbeans* pada JSF.

Untuk memperlihatkan bekerjanya transformasi XML, akan dibandingkan hasil luaran pada browser ketika servlet filter diaktifkan dan ketika servlet filter tidak diaktifkan. Berikut ini adalah luaran yang dihasilkan ketika mengakses halaman login **tanpa** mengaktifkan transformasi XML



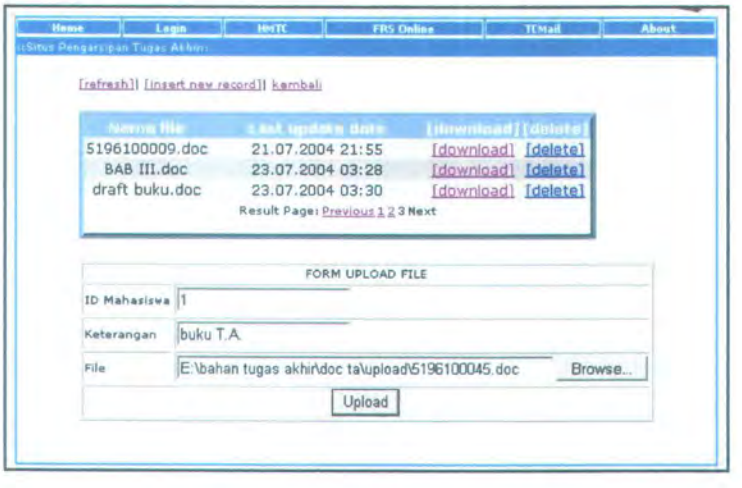
Gambar 5.7 Tampilan layar login tanpa transformasi XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
      <header>
            <center>Halaman Login</center>
      </header>

      <content>
            <center><B>Login Box</B>
            <br/>

      <form method="post" action="/plex03/faces/login.jsp">
      <table width="32%" border="1" align="center"
cellpadding="4" cellspacing="0">
        <tr>
          <td width="44%">Username</td>
          <td width="4%"><div align="center">:</div></td>
          <td width="52%">
      <input type="text" name="input_username"
            value="admin" style="color: blue"/>
            </td>
        </tr>
 . . .
```

Gambar 5.8 Kode luaran layar login tanpa transformasi XML

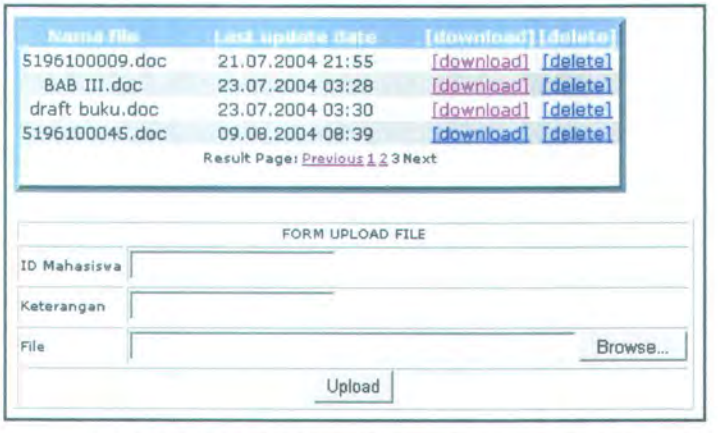
5.2.2. Skenario Kedua

Skenario kedua digunakan untuk menguji proses administrasi dokumen Tugas Akhir, yaitu untuk proses upload, dan download, serta delete dokumen yang tersimpan didatabase.Skenario ini diawali dengan mengisi form upload arsip T.A. seperti pada gambar berikut.



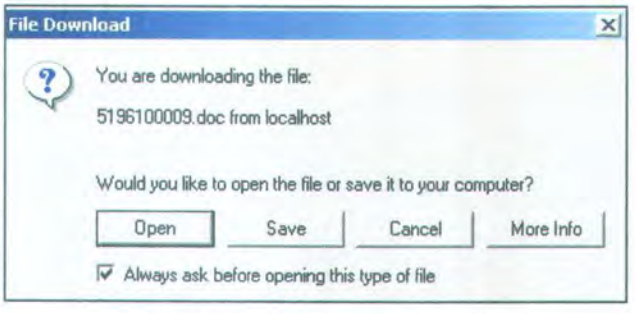
Gambar 5.9 Mengisi form upload

Setelah form ini di-*submit,* maka file yang telah dipilih disimpan di database. Untuk melihat file ini di dalam daftar arsip, terlebih dahulu diklik *link* refresh yang akan memuat kembali daftar file dari database. Gambar berikut memperlihatkan daftar file setelah proses *refresh* .



Gambar 5.10 Daftar file setelah melakukan *upload* file.

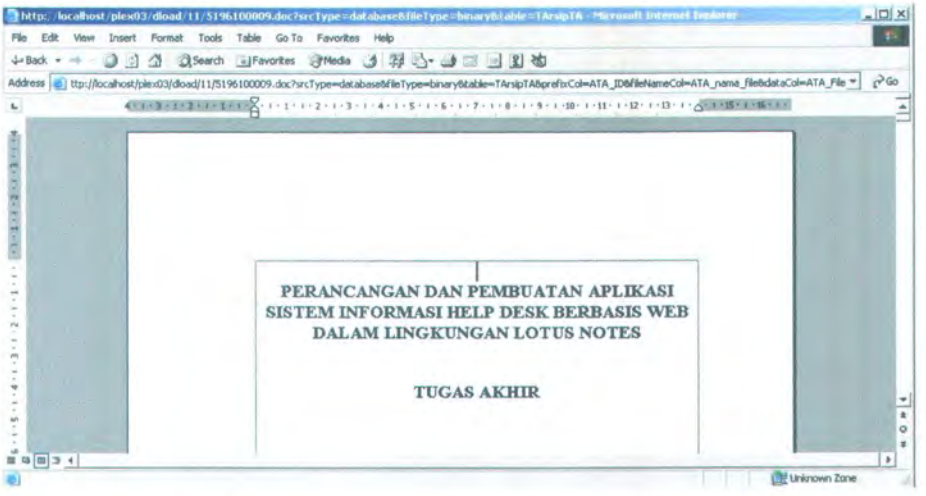
Untuk melakukan proses download file, dipilih file yang akan di-*download,* dilanjutkan dengan mengeklik link [download] pada daftar file arsip. Selanjutnya file akan di-*download.* Pada *browser* Internet Explorer, proses download file ini ditandai dengan munculnya kotak dialog File Download seperti pada gambar 5.11 berikut.



Gambar 5.11. Kotak dialog File Download

Jika tombol "save" diklik maka *browser* akan membuka kotak dialog *save file*. Jika tombol Open ini diklik, maka *browser* akan langsung membuka file yang

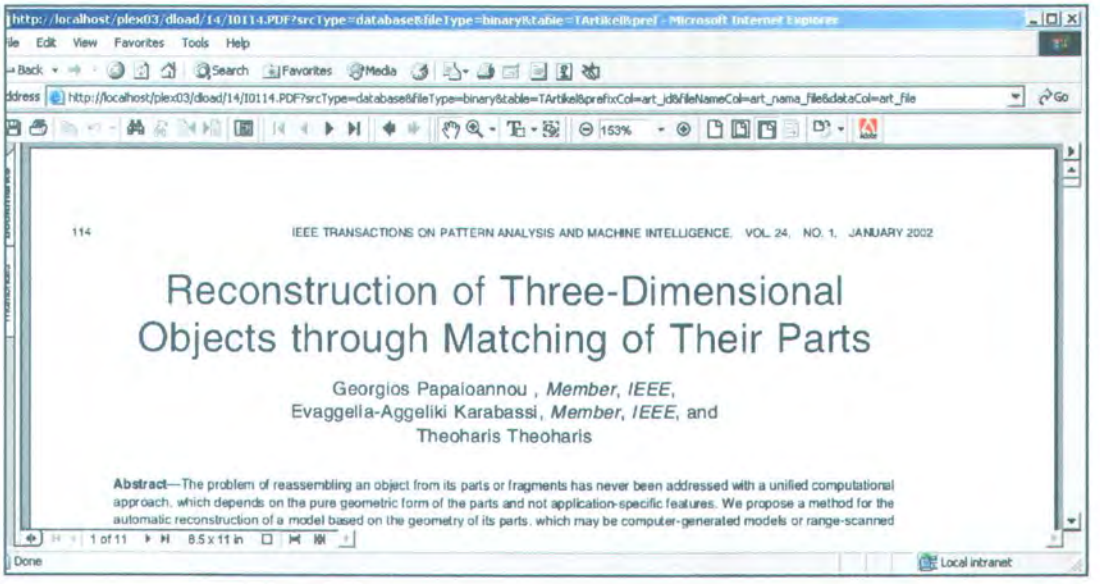
dipilih. Gambar 5.12 memperlihatkan *browser* yang membuka file Microsoft Word.



Gambar 5.12 Tampilan file Microsoft Word yang dibuka dari *browser*.

Pada arsip artikel juga dapat dilakukan proses *download* file. Terlebih dahulu dipilih file yang akan di-*download* dari daftar file artikel (Lihat gambar 5.13). Kemudian klik link [download]. Seperti pada arsip tugas akhir, *browser* akan membuka kotak dialog File Download(Lihat gambar 5.14). Jika tombol Open ini diklik, maka *browser* akan langsung membuka file yang dipilih. Gambar 6.15 memperlihatkan *browser* yang membuka sebuah file pdf.

Gambar 5.13. Daftar file arsip artikel.



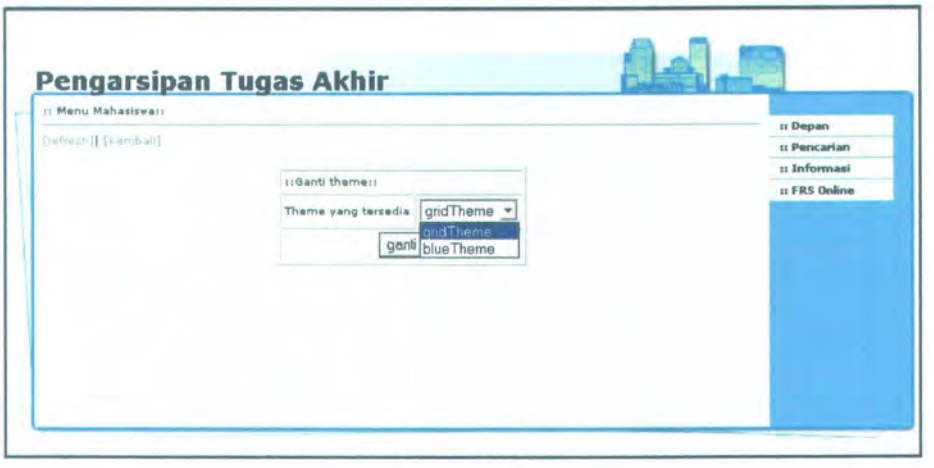
Gambar 5.14. Tampilan file pdf yang dibuka dari *browser*.

Menghapus file arsip dari database, dapat dilakukan dengan mengeklik *link* [delete] sesuai dengan baris file yang dipilih.

5.2.3. Skenario Ketiga

Skenario ketiga dilakukan untuk menguji fitur aplikasi mengubah *theme* tampilan. Pada aplikasi ini disediakan dua buah *theme,* yaitu *gridTheme* dan *blueTheme.* Pengguna dapat mengubah *theme* melalui menu yang disediakan.

Gambar 5.15 adalah tampilan menu theme ketika pengguna masuk ke bagian *ganti theme* dengan theme awal gridTheme. Pada tampilan ini pengguna mendapat dua pilihan *theme* yang dapat dipakai. Gambar 5.16 adalah tampilan ketika pengguna telah mengganti theme menjadi blueTheme.



Gambar 5.15. Tampilan menu theme.

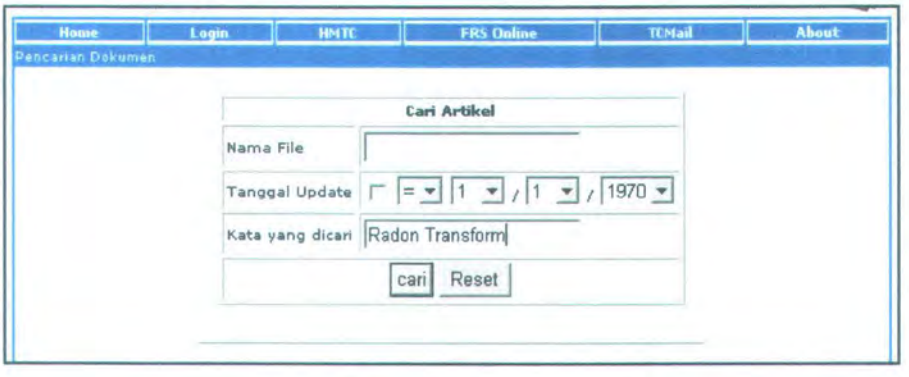
Gambar 5.16 Tampilan menu dengan blueTheme

5.2.4. Skenario Keempat

Skenario ini digunakan untuk menguji kemampuan dan kebenaran proses pencarian oleh aplikasi. Skenario ini dilakukan dengan memasukkan sebuah kata kunci dari dokumen tertentu ke fungsi pencarian aplikasi. Kemudian dari daftar dokumen hasil pencarian akan diperiksa apakah aplikasi dapat menemukan dokumen yang dicari.

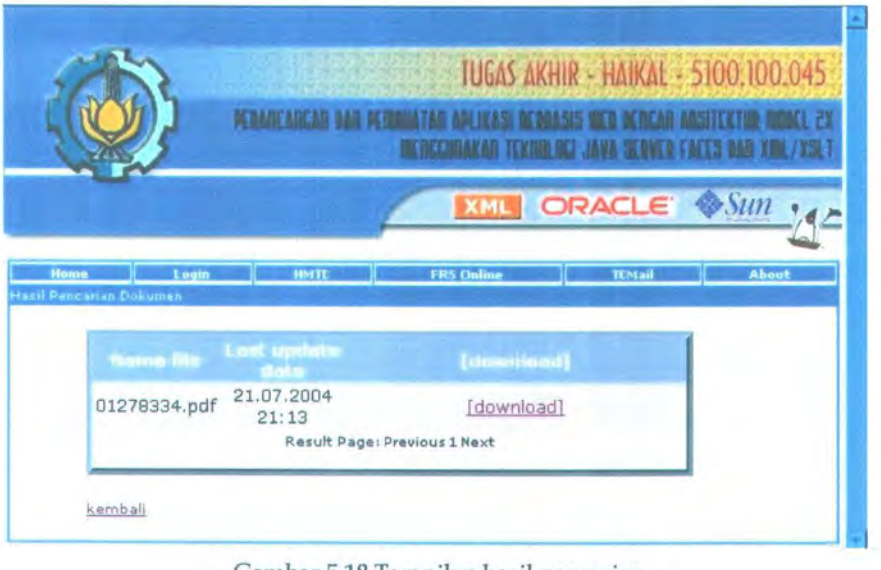
Untuk skenario ini, database diisi terlebih dahulu dengan sampel data pencarian. Sebagai sampel data, di-*upload* data berupa dokumen berformat pdf yang berasal dari jurnal IEEE sejumlah 108 file.

Sebagai kata kunci, dipilih kata "Radon Transform" yang terdapat pada file 01278334.pdf. file ini berisi artikel IEEE berjudul "Efficient Blind Image Restoration Using Discrete Periodic Radon Transform". Kata kunci diisikan ke dalam form pencarian, lalu form di-*submit.* Lihat gambar 5.17.



Gambar 5.17. Mengisi form pencarian.

Tampilan hasil dari pencarian dapat dilihat pada gambar 5.18. Dari daftar ini dapat diketahui bahwa pencarian telah berhasil.



Gambar 5.18 Tampilan hasil pencarian

5.3. EVALUASI

Dari uji coba yang telah dijalankan, aplikasi telah dapat mengimplementasikan arsitektur Model 2X, memanfaatkan teknologi JSF dan format data XML. Operasi-operasi database yang dijalankan oleh *formbean* dari JSF berjalan lancar. Proses pengubahan desain antarmuka atau layer presentasi juga berjalan sukses. Untuk manajemen dokumen dengan Oracle Text juga berjalan lancar. Terhadap database, dokumen dapat di-*download,* di-*upload,* dan dihapus.

Beberapa kelemahan terletak pada proses ketika membuka atau melakukan submit pada halaman JSP untuk yang pertama kali, yaitu lambatnya *run-time engine* memproses *request* dari *client.* Hal ini disebabkan karena web server harus melakuakan kompilasi terlebih dahulu terhadap halaman JSP yang baru dibuat. Kelemahan ini teratasi

BAB VI

KESIMPULAN DAN SARAN

BAB VI

KESIMPULAN DAN SARAN

Pada bab terakhir ini dijelaskan mengenai kesimpulan yang didapat dari Tugas Akhir ini dan saran-saran yang perlu diperhatikan untuk pengembangan perangkat lunak selanjutnya

6.1. KESIMPULAN

Dari keseluruhan pembuatan Tugas Akhir dapat diambil beberapa kesimpulan sebagai berikut:

1. Arsitektur Model 2X dapat diimplementasikan dalam bentuk aplikasi Pengarsipan Tugas Akhir sebagai sebuah aplikasi berbasis web dengan memanfaatkan teknologi JavaServer Faces sebagai struktur Model-View-Controller, Servlet Filter dan XML/XSLT sebagai komponen transformasi XML, serta Oracle Text untuk penyimpanan dan pencarian dokumen.
2. Integrasi teknologi JSF pada arsitektur Model 2X adalah sebagai struktur MVC dimana FacesServlet berperan sebagai Controller, formbean sebagai Model, dan JSP sebagai View. Untuk mendukung proses integrasi ini perlu dibuat sebuah renderkit khusus yang mendukung format XML sebagai keluarannya.

6.2. SARAN

Dari keseluruhan pengerjaan tugas akhir ini, berikut beberapa saran yang dapat diberikan penulis untuk perbaikan dan penyempurnaan selanjutnya:

1. Dengan proses transformasi XML yang lebih kompleks pada arsitektur model 2X dapat dikembangkan perangkat lunak berbasis web yang dapat berinteraksi dengan beragam perangkat lunak lain.
2. pengembangan lebih lanjut komponen antarmuka pada JSF akan menghasilkan proses pengembangan aplikasi yang lebih cepat dan modular.
3. Untuk proses manajemen dokumen yang lebih baik, pemanfaatan fitur-fitur Oracle Text yang belum terimplementasi pada tugas akhir ini (seperti GIST, HIGHLIGHT, MARKUP) akan menambah kualitas layanan dokumen.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

[BEL03] Bell, Jennifer, Eric Amstrong, "The Java Web Service Tutorial", SunMicrosystem, 2003

[BOO99] Booch, Rumbaugh, Jacobson, "Software Architecture Definition" , 1999, http://www.sei.cmu.edu/architecture/definitions.html.

[BRU03] Bruchez, Erik, Omar Tazi, "Integrating JSP/JSF and XML/XSLT: The Best of Both Worlds", 2003, http://www.theserverside.com.

[FIS02] Fischer, Tom, John Slater, Pete Stromquist, "Professional DesignPattern in VB.NET: Building Adaptable Applications", WroxPress, 2002.

[ORA02] Oracle Corporation, "Oracle Text Application Developer's Guides Release 9.2", 2002.

[RAT00] Rational Software Corporation, "Rational University Object Oriented Analysis and Design Using the UML", 2000.

[SUB01] Subrahmanyam, Allamaraju', Cedric Buest, John Davis, "Professional JavaServer Progamming J2EE 1.3 Edition", WroxPress, 2001.

[QUA03] Quatrani, Terry, *Visual Modeling with Rational Rose and UML*, Addison Wesley Longman, Inc., 2003.

LAMPIRAN

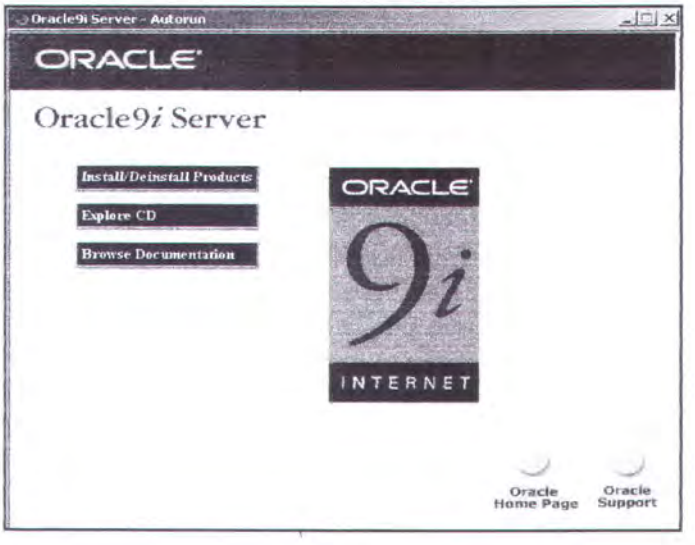
LAMPIRAN

Pada bagian ini diberikan lampiran beberapa panduan untuk instalasi perangkat lunak yang dibutuhkan untuk menjalankan aplikasi Pengarsipan Tugas Akhir ini.

A. PANDUAN INSTALASI ORACLE 9iR2 DATABASE

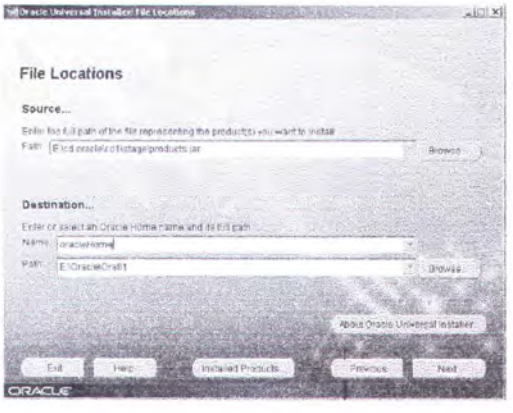
Langkah-langkah melakukan instalasi Oracle 9iR2:

1. Masukkan CD 1 dari 3 CD instalasi Oracle 9iR2. CD memiliki program autorun yang berjalan secara otomatis bila CD dimasukkan.

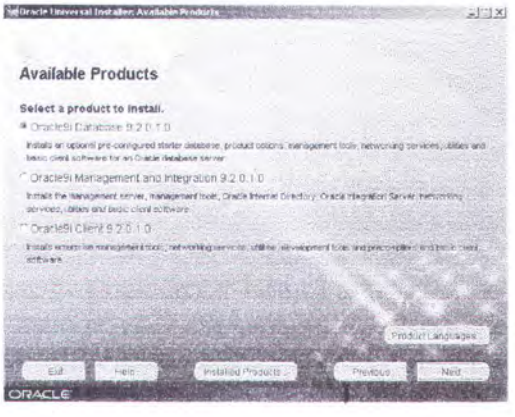


2. Klik **install/Deinstall** Product. Dialog berikutnya memberikan opsi untuk melihat produk Oracle yang tekah terinstal di komputer lokal. Klik **Next** untuk melanjutkan instalasi.

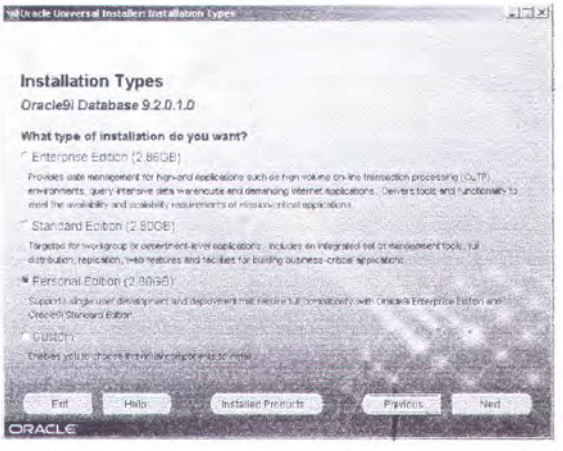
3. Dialog selanjutnya memungkinkan untuk menentukan lokasi folder instalasi database Oracle, serta nama *Oracle Home*. Setelah mengisi nilai *setting* tersebut, klik **Next** untuk melanjutkan instalasi.



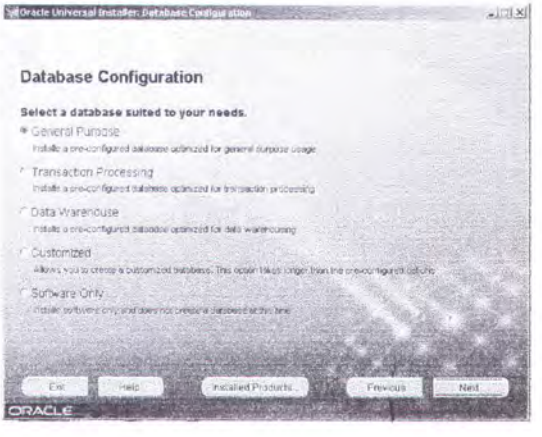
4. Dialog selanjutnya memberikan pilihan setup, pilih yang pertama (instalasi database), lalu klik **Next.**



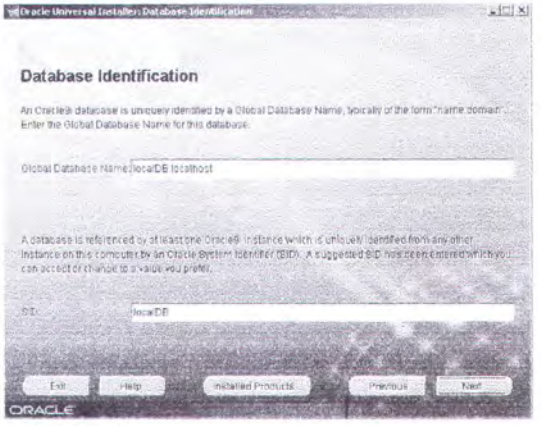
5. Untuk dialog berikutnya terdapat pilihan tipe instalasi database. Fitur Oracle Text terdapat pada ketiga tipe tersebut, sehingga pemakai tetap dapat menggunakan Oracle Text untuk semua tipe database yang diistal. Disini Penulis menggunakan tipe Personal Edition. Selanjutnya klik **Next.**



6. Kemudian untuk dialog Database Configuration, pilih General Purpose Database.

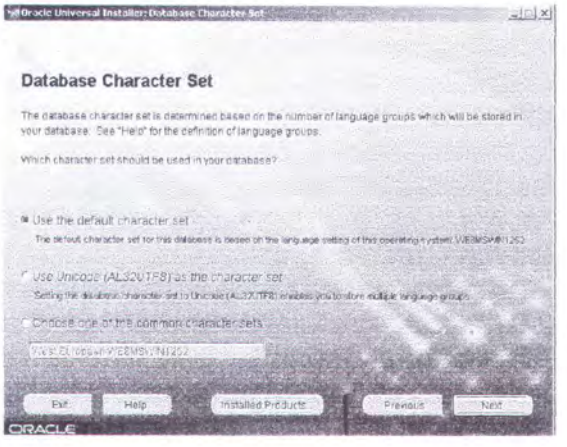


7. Dialog selanjutnya adalah untuk mengisi setting ID database, yaitu Global Database Name serta System Identifier.

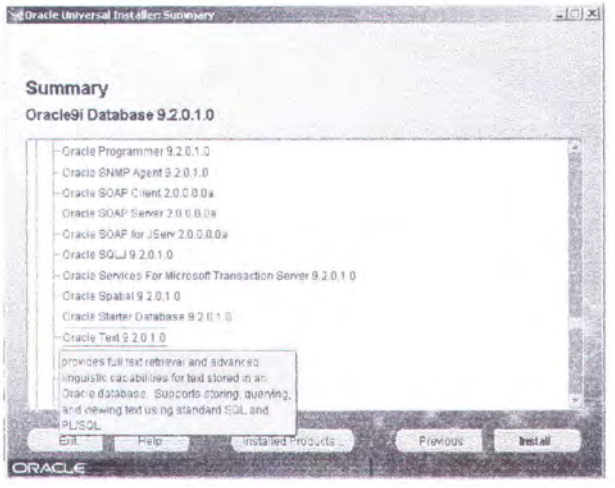


8. Kemudian untuk dialog Database File Location, pilih folder penyimpanan data dari Oracle.

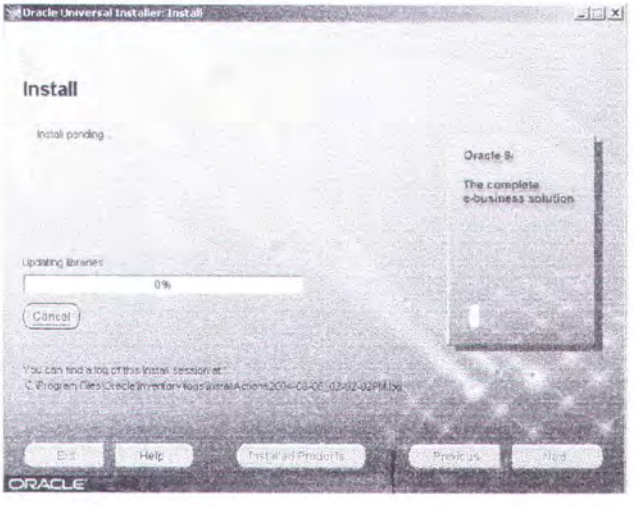
9. Berikutnya menentukan set karakter yang digunakan. Ini dibutuhkan jika menggunakan bahasa yang memerlukan karakter khusus di luar huruf latin.



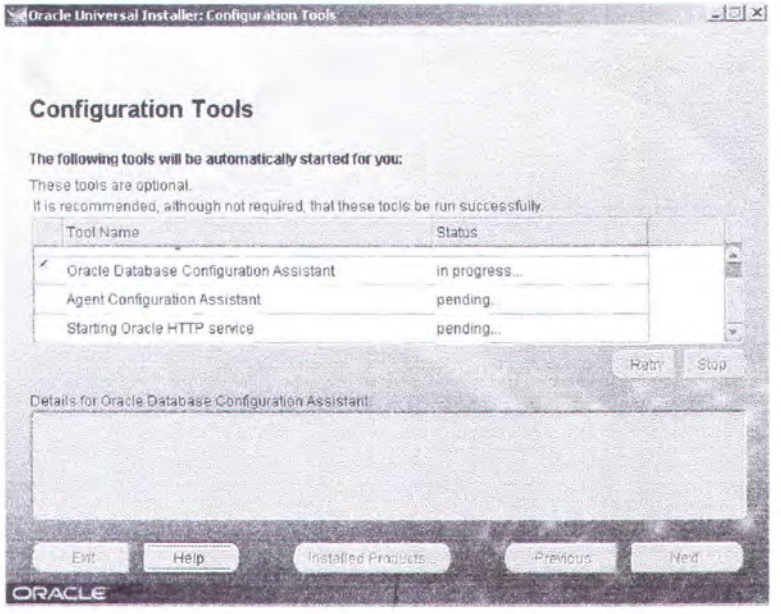
10. Dialog berikutnya adalah konfirmasi setting keseluruhan yang telah dimasukkan Sebelum instalasi dilanjutkan. Perhatikan bahwa **Oracle Text** telah dimasukkan di sini. Klik **Install** untuk melanjutkan instalasi database.



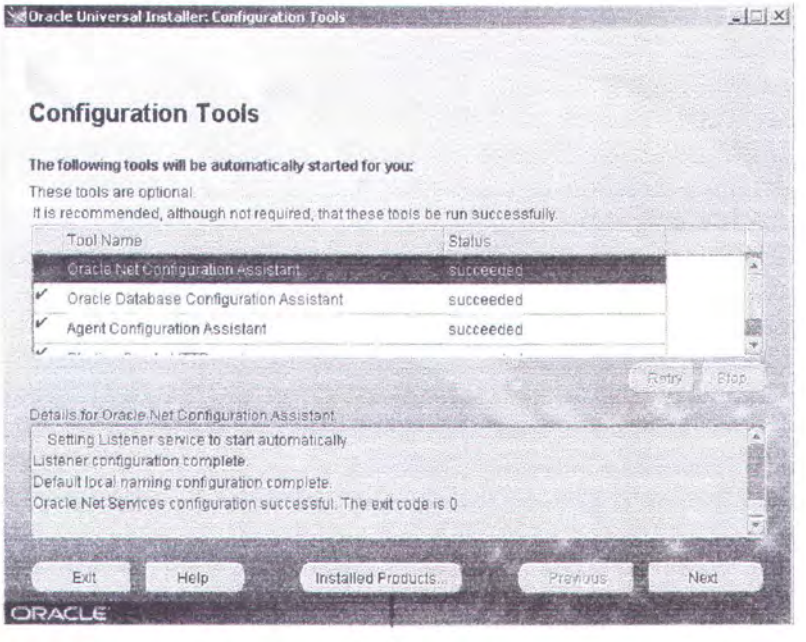
Selanjutnya proses instalasi dijalankan dengan ditandai *progress bar* pada dialog instalasi.



11. Setelah proses ini selesai, Oracle Installer akan melakukan konfigurasi terhadap database.



12. Proses instalasi berakhir dengan selesainya konfigurasi ini. Klik Next jika akan menginstal tool Oracle lainnya. Klik Exit untuk keluar dari instalasi.



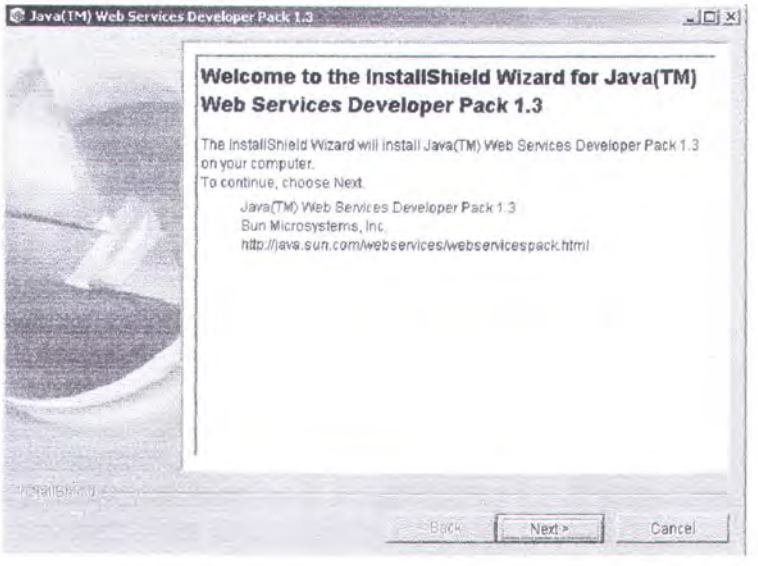
B. PANDUAN INSTALASI JAVA WEB SERVICE DEVELOPER PACK 1.3 (JWSDP 1.3)

Untuk memperoleh paket instalasi JWSDP, *download* dari alamat web: http://java.sun.com/webservices/downloads/webservicespack.html. Pada paket ini sudah disertakan *library* JavaServer Faces (JSF). Untuk memperoleh paket JSF secara terpisah, dapat di-*download* dari alamat web: http://java.sun.com/j2ee/javaserverfaces/downloads/index.html.

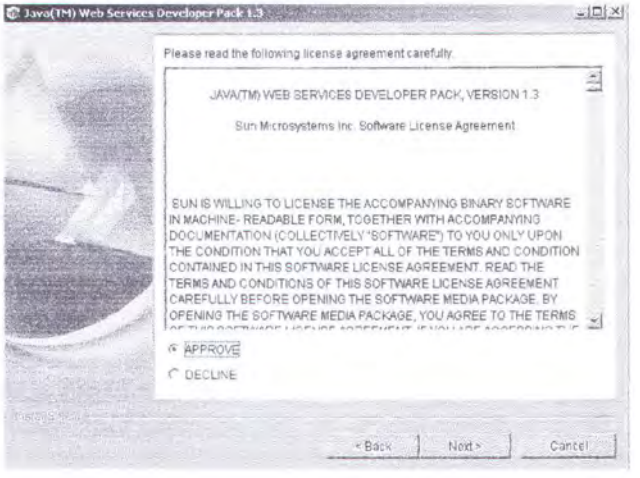
Instalasi JWSDP mensyaratkan adanya instalasi Java Developer Kit (JDK) versi 1.4.1 atau yang di atasnya. Untuk memperoleh paket instalasi JDK dapat di-*download* dari alamat web: http://java.sun.com/j2se.

Berikut ini petunjuk instalasi JWSDP 1.3.

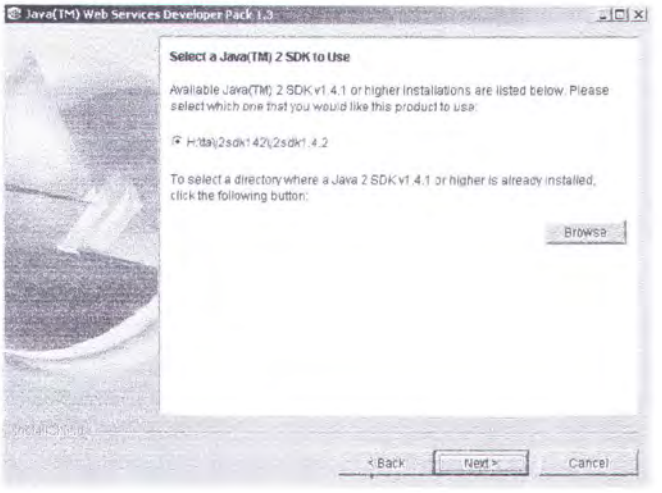
1. Klik ganda pada file instalasi yang telah di-*download*. Berikut dialog yang pertama kali tampil untuk proses instalasi.



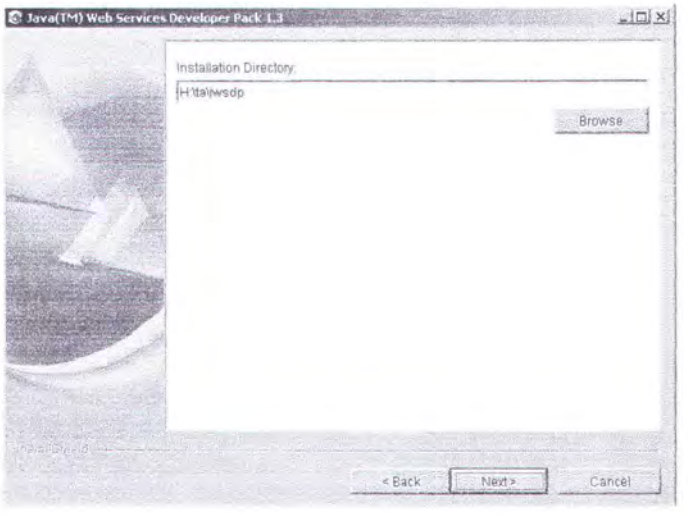
2. Selanjutnya tampil dialog persetujuan lisesnsi. Pilih **Agree,** lalu klik **Next** untuk melanjutkan instalasi.



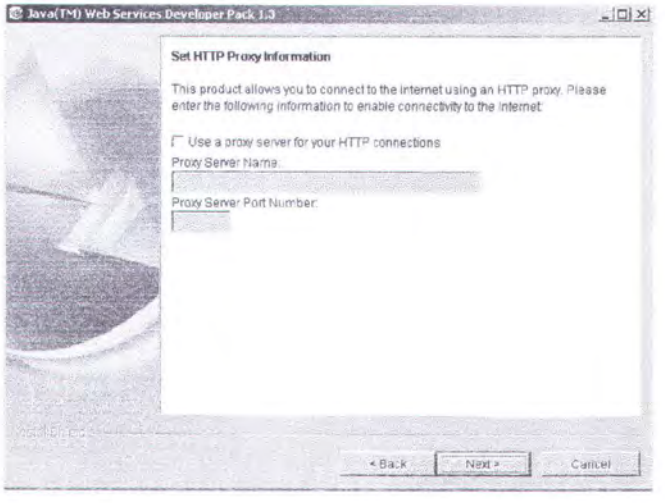
3. Selanjutnya tampil dialog memilih instalasi Java Developer Kit(JDK) yang akan digunakan. JWSDP 1.3 mensyaratkan instalasi JDK versi 1.4x untuk instalasi. Jika instalasi JDK tidak ditemui diantara pilihan yang disediakan, klik tombol **browse** untuk memilih instalasi JDK pada *file system.* Kemudian klik **Next** untuk melanjutkan instalasi.



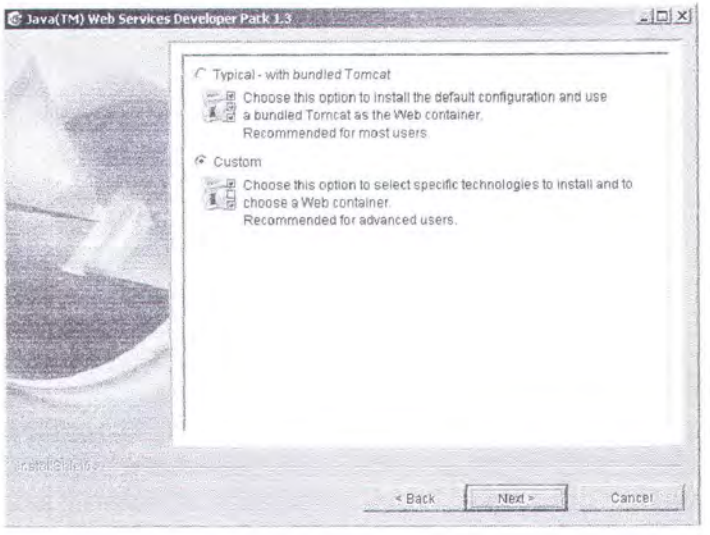
4. Dialog berikutnya adalah memilih folder instalasi JWSDP. Setelah menentukan lokasi folder instalasi, klik **Next** untuk melanjutkan instalasi.



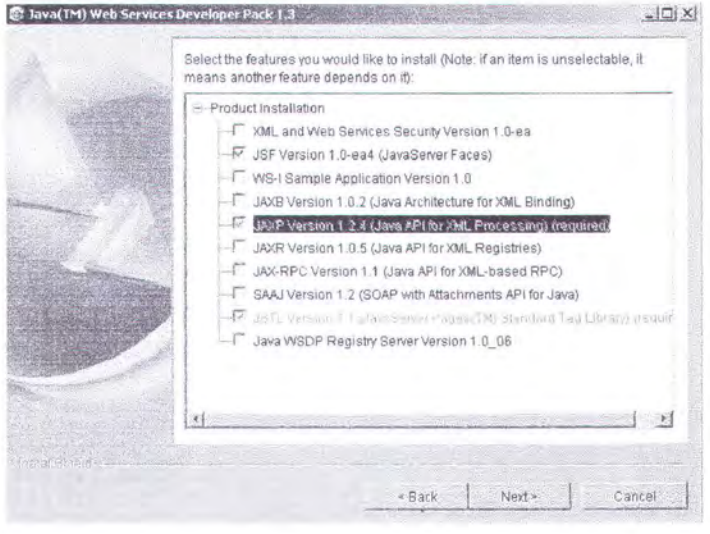
5. Dialog selanjutnya adalah menentukan setting proxy server untuk koneksi ke internet. Opsi ini dipilih jika tersedia koneksi ke internet yang melalui server proxy.



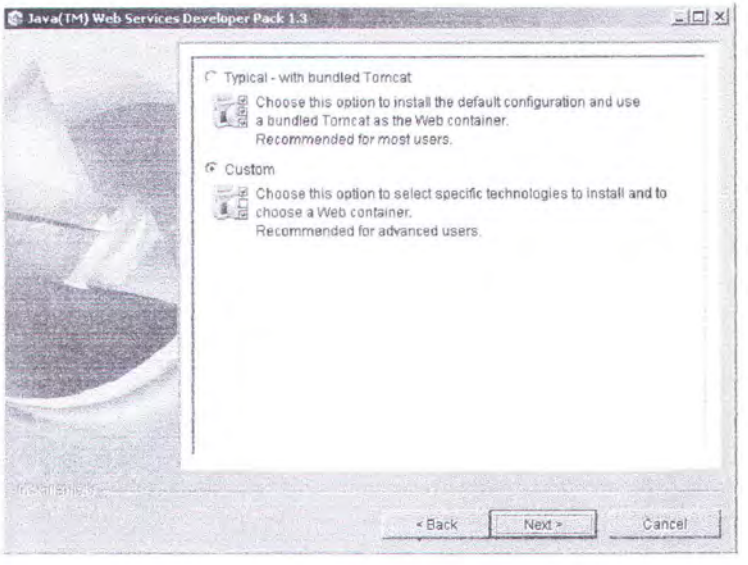
6. Dialog selanjutnya memilih tipe instalasi. Pilih tipe **Custom** agar dapat memilih instalasi khusus JSF, lalu klik **Next** untuk melanjutkan instalasi.



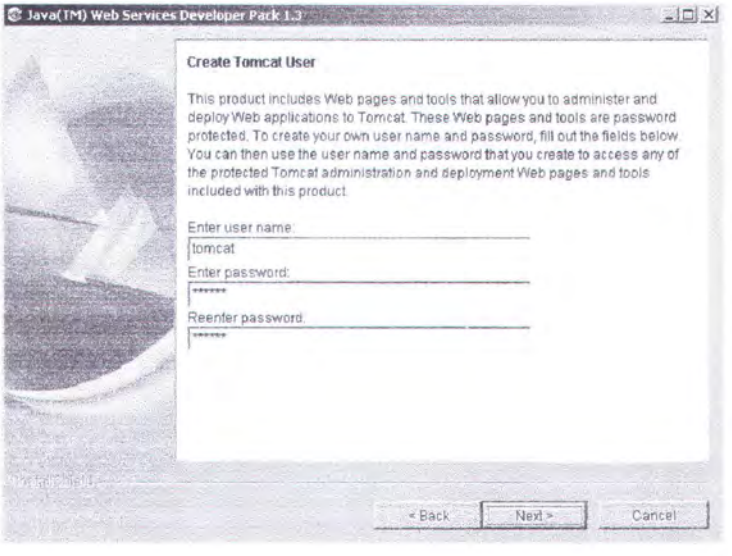
7. Kemudian pada dialog berikutnya pilih produk instalasi yang dibutuhkan. Untuk penggunaan JSF, hanya diperlukan produk JSF, JAXP(Java API for XML Processing), serta JSTL.(JSP Standard Tag Library), klik **Next** untuk melanjutkan instalasi.



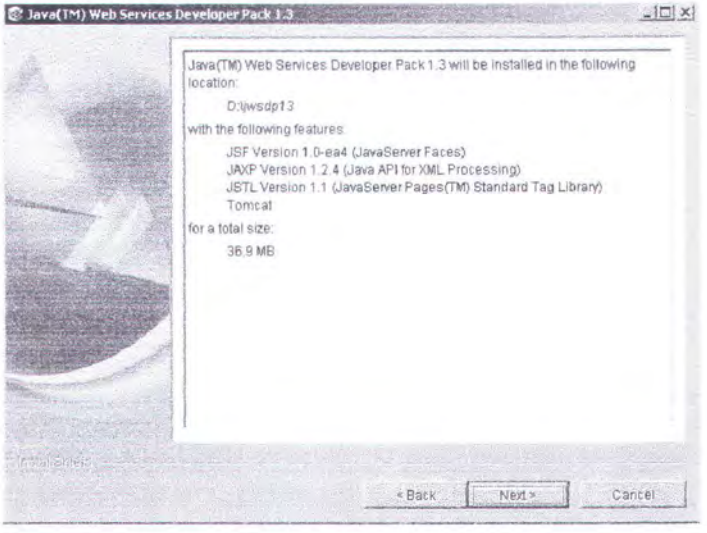
8. Dialog berikutnya memberikan pilihan apakah turut menginstal *web container* Tomcat atau tidak. pilih opsi pertama (Tomcat *included)* jika belum memiliki *web container.*



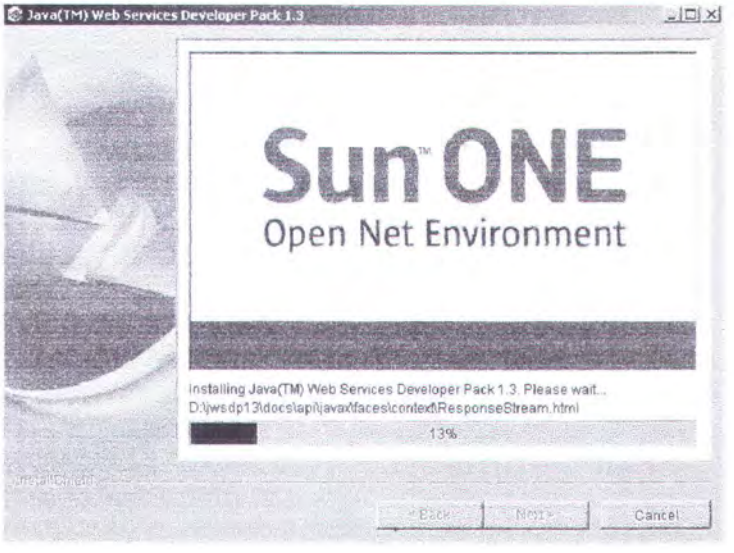
9. Dialog berikutnya adalah menentukan *username* dan *password* administator *web container manager* yang diinstal.



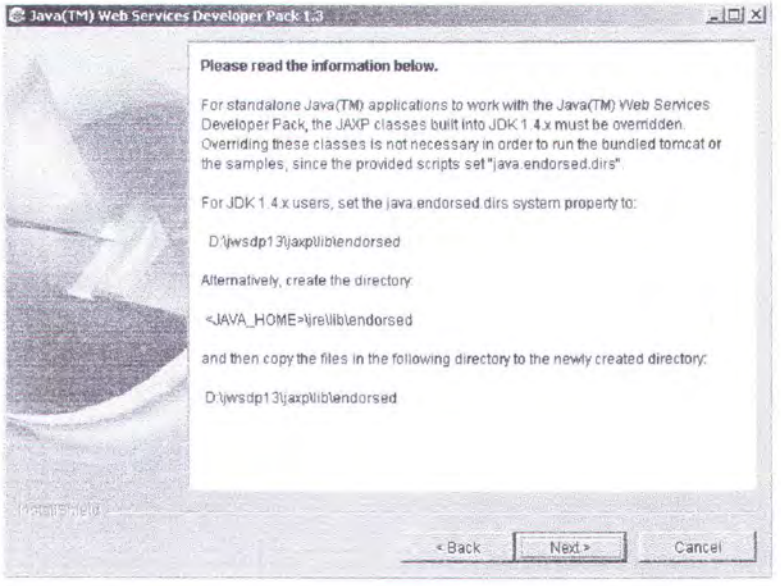
10. Kemudian diberikan konfirmasi setting yang telah ditentukan sebelum instalasi JWSDP 1.3 dilanjutkan. Klik **Next** untuk melanjutkan instalasi.



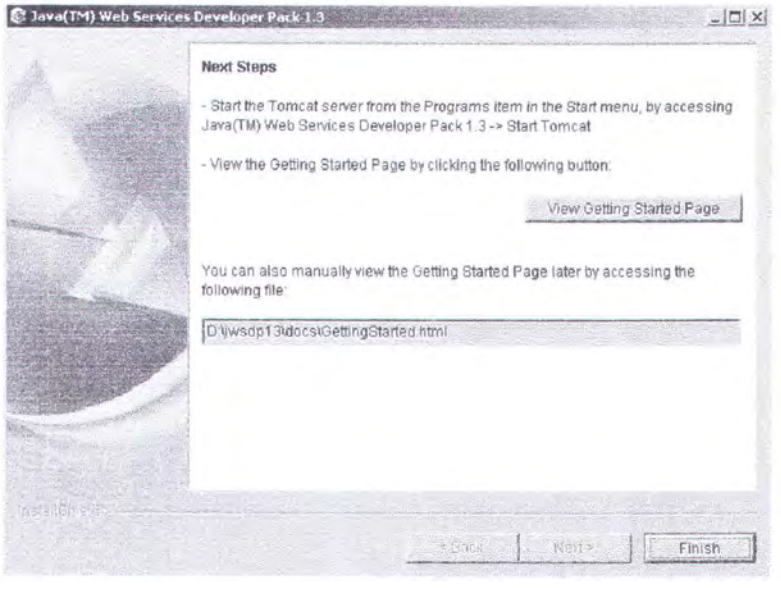
11. Berikutnya instalasi JWSDP dijalankan dengan ditandai *progress bar.*



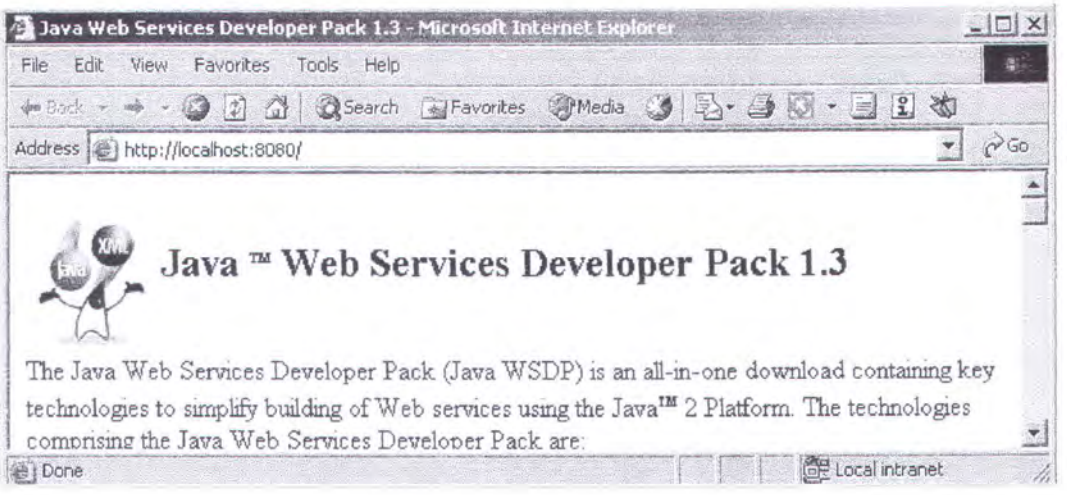
12. Setelah instalasi selesai ditampilkan informasi konfigurasi yang diperlukan jika JWSDP akan digunakan untuk pengolahan XML pada aplikasi yang berdiri sendiri. Klik **Next** untuk melanjutkan.



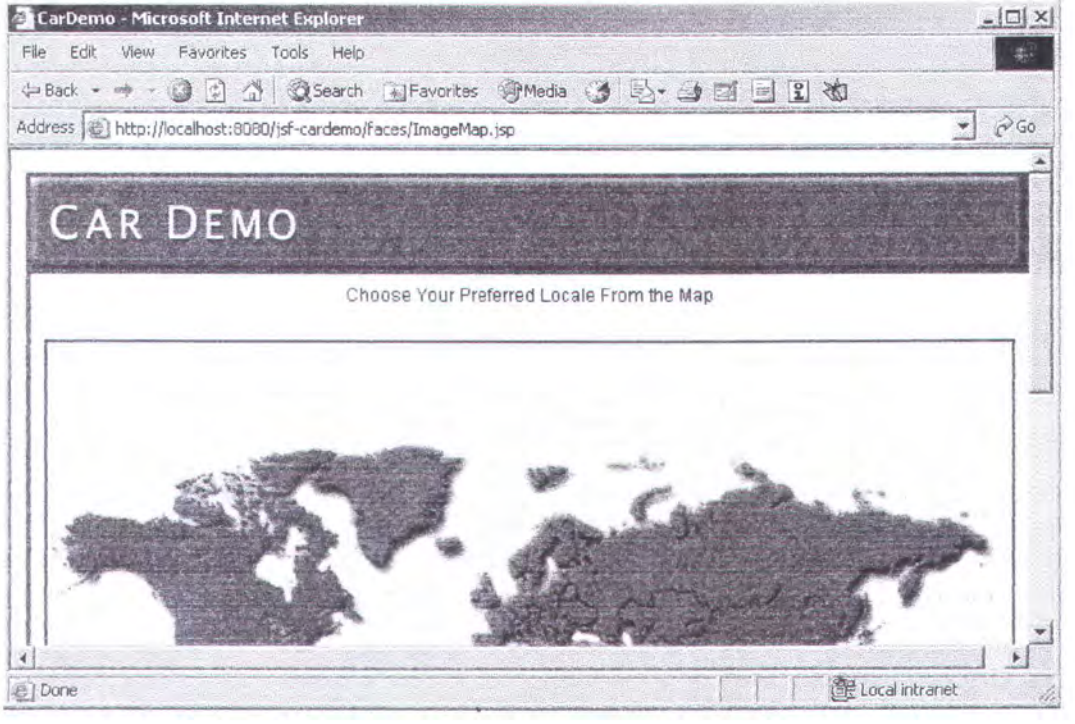
13. Dialog berikutnya berisi informasi untuk halaman web *getting started* dari JWSDP 1.3. Dialog ini menandai selesainya proses instalasi JWSDP 1.3 Klik **finish** untuk keluar.



14. Untuk melihat hasil instalasi JWSDP, buka halaman web http://localhost:8080/ . Akan ditampilkan alamat web berikut:



15. Untuk melihat halaman web demo framwork JSF, buka alamat web: http://localhost:8080/jsf-cardemo/. Akan ditampilkan alamat web berikut:



C. PANDUAN INSTALASI APLIKASI PENGARSIPAN TUGAS AKHIR

Pada bagian ini akan dijelaskan hal-hal yang perlu disiapkan sebelum menjalankan aplikasi Pengarsipan Tugas Akhir. Secara Garis besar, langkah-langkahnya adalah menyiapkan database beikut indeks Oracle Text, serta menginstal aplikasi pada *web container.*

Setelah database Oracle terinstal, selanjutnya membuat tabel-tabel yang dibutuhkan, beserta indeks Oracle Text. Membuat tabel dan indeks dapat dilakukan dengan mengeksekusi *script* SQL yang telah diberikan pada bab 4.1.

Langkah-langkah yang perlu dilakukan untuk menginstal aplikasi pada *web container* adalah:

1. *Copy* file aplikasi Pengarsipan Tugas Akhir ke folder `webapps` pada *web container.* gambar berikut adalah struktur folder pada *web container* JWSDP 1.3, aplikasi diinstal pada folder `pta`.

Struktur folder aplikasi web

2. Menyiapkan file-file konfigurasi aplikasi, yaitu file `pta-app-config.xml` dan `pta-pipe-config.xml`, keduanya diletakkan pada folder `pta/WEB-INF`.

   Pada file `pta-app-config` nilai setting yang perlu diisi adalah:

   - ✓ Elemen `app-name` dan `app-desc`, berisi nama dan deskripsi aplikasi.
   - ✓ Elemen `database`, berisi nama (untuk identifikasi pada aplikasi), URL database (dengan format url JDBC), username, serta password ke database.
   - ✓ Elemen `uagent-mapping`, berisi konfigurasi *device* yang di-*support* aplikasi. Setting konfigurasi ini adalah untuk pengembangan aplikasi selanjutnya. Pada saat buku ini ditulis, aplikasi ini hanya mendukung device HTML *browser* (misalnya Internet Explorer dan Netscape Navigator).
   - ✓ Elemen `admin-login` berisi username dan password untuk pengguna Administrator aplikasi.

   Pada file `pta-pipe-config.xml`, nilai setting yang perlu diisi adalah:

   - ✓ Elemen `pipe-name`, berisi nama konfigurasi ini.
   - ✓ Elemen `process-param`, berisi konfigurasi proses transformasi xml yang dilakukan aplikasi. Elemen ini memiliki sub Elemen:
     1. Elemen `name`, nama transformasi xml.
     2. Elemen `param-name`, nama parameter yang akan dipakai untuk proses transformasi ini.
     3. Elemen `param-value`, berisi nilai-nilai parameter yang dapat dipakai. Sebuah elemen `process-param` dapat memiliki lebih dari

satu `param-value`, bergantung pada jumlah variasi *template* transformasi yang dimiliki oleh proses transformasi ini (berupa file xsl).

Elemen `param-value` ini juga berasosiasi dengan file xsl, di mana sebuah elemen `process-value` sesuai dengan nama sebuah file transformasi xsl tanpa ekstensi xsl.

Berikut ini contoh file konfigurasi pta-app-config.xml dan pta-pipe-config.xml.

```
<application-settings>
 <app-name> FPR </app-name>
 <app-desc> Final Project Repository </app-desc>
 <database>
        <name>MyDB-1</name>
        <driver>oracle.jdbc.OracleDriver</driver>
        <url>jdbc:oracle:thin:@127.0.0.1:1521:PRIMADB</url>
        <user>pta</user>
        <password>ptapwd</password>
<uagent-mapping>
        <uagent-key>Mozilla</uagent-key>
        <uagent-suffix>html</uagent-suffix>
        <uagent-content-type>text/html</uagent-content-type>
 </uagent-mapping>

 <admin-login>
        <username>admin</username>
        <password>zx</password>
 </admin-login>
</application-settings>
```

contoh file pta-app-config.xml

```
<pipe-config>
        <pipe-name>pipe01</pipe-name>
<process-param>
        <name>coloring</name>
        <param-name>color</param-name>
        <param-value>login_c_blue</param-value>
        <param-value>login_c_yellow</param-value>
</process-param>
<process-param>
        <name>theme-config</name>
        <param-name>theme</param-name>
        <param-value>common-html</param-value>
        <param-value>green-html</param-value>
        <param-value>blue-html</param-value>
</process-param>
</pipe-config>
```


contoh file pta-pipe-config.xml

3. Menyiapkan setting pada file `web.xml`. Nilai setting yang perlu diisikan adalah:

   - Setting lokasi file konfigurasi aplikasi (pada elemen `context-param`):
     - ✓ `AppConfigFile` (lokasi `file pta-app-config.xml`).
     - ✓ `AppRuleFile` (lokasi file `digester-rule-app.xml`).
     - ✓ `PipeConfigFile`(lokasi file `pta-pipe-config.xml`).
     - ✓ `PipeRuleFile` (lokasi file digester-rule-pipe.xml).
     - ✓ `xslPath` (lokasi folder file xsl).
   - Servlet dan Servlet Filter untuk aplikasi:
     - ✓ Servlet yang diperlukan aplikasi untuk proses *download* (oc4jdownload) dan Servlet yang memuat konfigurasi (SettingsServlet) aplikasi, diisi pada elemen servlet dan servlet-mapping.
     - ✓ Servlet Filter untuk proses transformasi XML (XSLFilter), diisi pada elemen filter dan filter mapping.
   - Untuk konfigurasi JavaServer Faces, nilai-nilai yang perlu diisi adalah:
     - ✓ Konfigurasi parameter JSF: `saveStateInClient`, `javax.faces.application.CONFIG_FILES`, `com.sun.faces.validateXml` pada elemen `context-param`.
     - ✓ Kelas listener JSF, pada elemen listener, yaitu `com.sun.faces.config.ConfigListener`.
     - ✓ Servlet Controller JSF (Faces Servlet), diisi pada elemen servlet dan servlet-mapping.

Berikut ini contoh file `web.xml` yang telah dikonfgurasi.

```
<web-app>
    <description>Aplikasi Pengarsipan Tugas Akhir</description>
    <display-name>PTA web app</display-name>
        <!-- Setting lokasi file konfigurasi aplikasi  -->
    <context-param>
              <param-name>AppConfigFile</param-name>
              <param-value>WEB-INF/pta-app-config.xml</param-value>
    </context-param>
     <context-param>
              <param-name>AppRuleFile</param-name>
              <param-value>WEB-INF/digester-rule-app.xml
                    </param-value>
       </context-param>
     <context-param>
              <param-name>PipeConfigFile</param-name>
              <param-value>WEB-INF/pta-pipe-config.xml</param-value>
    </context-param>
     <context-param>
              <param-name>PipeRuleFile</param-name>
              <param-value>WEB-INF/digester-rule-pipe.xml
       </param-value>
   </context-param>
     <context-param>
              <param-name>xslPath</param-name>
              <param-value>/WEB-INF/xsl/</param-value>
   </context-param>
   <!-- Konfigurasi parameter JSF -->
        <context-param>
         <param-name>saveStateInClient</param-name>
         <param-value>false</param-value>
     </context-param>
     <context-param>
         <param-name>javax.faces.application.CONFIG_FILES
                </param-name>
         <param-value>/WEB-INF/faces-config.xml</param-value>
     </context-param>
     <context-param>
         <param-name>com.sun.faces.validateXml</param-name>
         <param-value>true</param-value>
     </context-param>
        <!--    Kelas listener JSF -->
     <listener>
         <listener-class>
              com.sun.faces.config.ConfigListener
          </listener-class>
     </listener>
     <!-- Faces Servlet -->
     <servlet>
         <servlet-name>Faces Servlet</servlet-name>
         <servlet-class>javax.faces.webapp.FacesServlet
                </servlet-class>
         <load-on-startup> 1 </load-on-startup>
     </servlet>
<!-- Servlet untuk setting aplikasi dan proses download -->
     <servlet>
         <servlet-name>Setting-Servlet</servlet-name>
         <servlet-class>settings.SettingServlet</servlet-class>
         <load-on-startup>2</load-on-startup>
     </servlet>
```

```
    <servlet>
        <servlet-name>oc4jdownload</servlet-name>
        <servlet-
class>oracle.jsp.webutil.fileaccess.DownloadServlet</servlet-class>
    </servlet>
    <!-- Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Setting-Servlet</servlet-name>
        <url-pattern>/setting</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>oc4jdownload</servlet-name>
        <url-pattern>/dload/*</url-pattern>
    </servlet-mapping>
       <!-- Servlet Filter transformasi XML -->
   <filter>
              <filter-name>XSLFilter</filter-name>
              <display-name>filter</display-name>
              <filter-class>filter.XSLFilterParam</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>XSLFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
  </filter-mapping>
</web-app>
```

Contoh file web.xml yang telah dikonfigurasi