BAB III

LANDASAN TEORI

3.1. Teori Umum

Terdapat beberapa teori umum yang digunakan dalam implementasi web user management seperti yang diuraikan oleh definisi-definisi berikut.

3.1.1. CDM (Conceptual Data Model)

Merupakan model yang universal dan dapat menggambarkan semua struktut logic database (DBMS), dan tidak bergantung dari software atau pertimbangan struktur data storage. Sebuah CDM dapat diubah langsung menjadi PDM (Jogiyanto, 1990).

Aturan CDM sebagai berikut:

a. Mempresentasikan pengorganisasian data dalam format grafis.
b. Memverifikasi validasi desain data
c. Menghasilkan PDM dimana menspesifikasikan implementasi secara fisik pada *database*.

3.1.2. PDM (Physical Data Model)

Merupakan model ERD yang telah mengacu pada pemilihan software DBMS yang spesifik. Hal ini sering kali berbeda dikarenakan oleh struktur database yang bervariasi, mulai dari model schema, tipe data penyimpanan, dan sebagainya (Jogiyanto, 1990). PDM mengikuti aturan-aturan sebagai berikut:

a. Mempresentsaikan pengorganisasian data secara fisik dalam format grafis.

b. Menghasilkan script pembuat dan pemodifikasi *database*.

c. Mendefinisikan referential integrity *triggers and constraints*

Ada beberapa derajat relasi yang dapat terjadi, yaitu:

a. One to One Relationship

Menggambarkan bahwa antara satu *entity* hanya dapat berhubungan dengan satu *entity.* Biasanya derajat relasi ini digambarkan dengan simbol 1-1.

b. One to Many Relationship

Menggambarkan bahwa satu *entity* dapat memiliki hubungan dengan lebih dari satu entity. Biasanya derajat relasi ini digambarkan dengan simbol 1-N.

c. Many to Many Relationship

Menggambarkan bahwa lebih dari satu *entity* dapat memiliki hubungan dengan lebih dari satu *entity*. Biasanya derajat relasi ini digambarkan dengan simbol N-N.

3.1.3. Database

Menurut Jogiyanto (1999, p217), Basis data (*database*) adalah kumpulan dari data yang saling berhubungan satu dengan yang lainnya, tersimpan dalam perangkat keras computer dan digunakan oleh perangkat lunak untuk memanipulasinya.

Berikut ini adalah sifat-sifat basis data:

1. Berbagi Data

Data yang disimpan dalam basis data tidak secara umum digunakan oleh seseorang. Suatu basis data secara normal diharapkan bisa diakses oleh lebih dari satu orang, dan mungkin pada waktu yang sama.

2. Integrasi Data

Salah satu bentuk tanggung jawab pemakaian basis data yang utama adalah memastikan bahwa data terintegrasi. Suatu basis data harus menjadi koleksi data agar tidak terjadi redundansi data (yang berlebihan). Nilai data dikatakan redundansi bila suatu atribut memiliki dua atau lebih nilai yang sama.

3. Integritas Data

Tanggung jawab lain yang timbul sebagai konsekuensi dari data bersama adalah bahwa basis data perlu menunjukkan integritas. Dengan kata lain, basis data perlu secara akurat mencerminkan seluruh bidang yang mencoba model.

4. Keamanan Data

Salah satu cara untuk memastikan integritas basis data adalah dengan melakukan pembatasan akses yaitu pengamanan basis data.

5. Abstraksi Data

Suatu basis data dipandang sebagai model nyata. Informasi yang disimpan dalam basis data pada umumnya merupakan sebuah usaha untuk menyajikan sifat dari beberapa objek sesungguhnya. Oleh karena itu, suatu basis data adalah suatu abstraksi dari dunia nyata.

6. Independensi Data

Salah satu konsekuensi dari abstraksi adalah gagasan untuk *buffering data* dari proses yang menggunakan data.

3.2. Teori-teori Khusus (Peter, 2010)

Terdapat beberapa teori khusus yang digunakan dalam implementasi web user management seperti yang diuraikan oleh definisi-definisi berikut.

3.2.1. Pengertian Oracle ADF

Oracle Application Development Framework (Oracle ADF) adalah sebuah *framework* pengembangan aplikasi Java Platform, Enterprise Editon (Java EE) yang inovatif namun matang yang dikeluarkan oleh Oracle. Oracle ADF ini didukung secara langsung oleh IDE (*Integrated Development Environment*) yang telah memenangkan banyak penghargaan, yakni Oracle JDeveloper 11g.

Oracle ADF mengintegrasikan campuran subframework untuk memberikan fungsi tombol sebagai *object-relational-mapping* dan berbagai bentuk *service acces*, *data bindings*, dan *user interface*, bersama dengan *functional glue* untuk menahan itu semua bersama-sama.

3.2.2. Arsitektur Oracle ADF

Oracle ADF mengimplementasikan MVC, namun Layer Model terpisah dari Business Service.

Arsitektur Oracle ADF terdiri dari 4 layer:

a) *Layer Model:* menghubungkan *Business Service* dengan objek yang digunakan di *layer* lain. Implementasi *layer model* terletak di atas *layer Business Service*, yang menyediakan suatu *interface* tunggal sehingga memungkinkan *layer View* dan *Controller* mengakses *Business Service* yang beragam dengan cara yang konsisten.

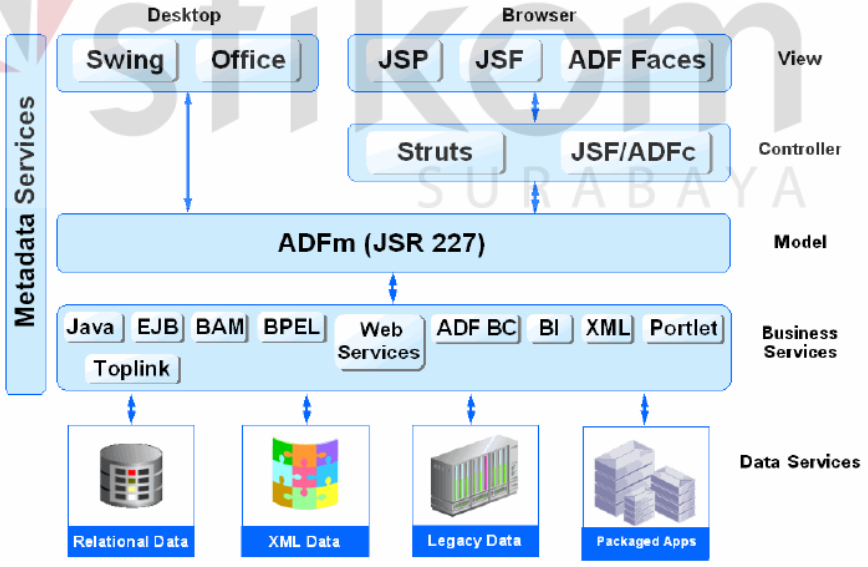
b) *Layer Model* sendiri terdiri dari dua komponen, yakni *data controls yang* mengabstraksi detail implementasi *Business Service*, termasuk informasi *properties*, *methods*, dan *type* yang dipakai dan *data bindings* yang

mengekspos *methods* dan *attributes Data Control* ke komponen UI, yang memberikan pemisah antara *View* dan *Model*.

c) *Layer Business Services*: menyediakan akses kedata dari berbagai sumber, *object* atau *relational mapping* dan menangani *business logic*. *Layer Business Services* di Oracle ADF bisa diimplementasikan dengan pilihan sebagai berikut: *class* Java, EJB 2.1/3.0, Web Services, JPA *objects*, dan Oracle ADF Business Components.

d) *Layer Controller:* menyediakan mekanisme untuk mengontrol *task flows* di aplikasi *web*. Ada 3 pilihan *controller* di JDeveloper: Oracle ADF Controller, JSF Controller atau Apache Struts.

e) *Layer View:* merepresentasikan UI aplikasi. *Layer View* bisa berupa *web client*, aplikasi *desktop client-server* atau implementasi *wireless* untuk *mobile device* seperti *handphone*.



Gambar 3.1. Arsitektur Oracle ADF

Oracle ADF memberikan pilihan kepada *developer* dalam menentukan jenis teknologi yang diimplementasikan pada setiap *layer*. Gambar di atas menunjukan pilihan-pilihan yang bisa dipakai oleh *developer* ketika membangun suatu aplikasi Oracle ADF. EJB, Web Services, JavaBeans, dan JPA/EclipseLink/TopLink bisa dipakai sebagai *Business Services* untuk Oracle ADF. Yang termasuk *layer View* antara lain aplikasi Swing, integrasi dengan MS Office, maupun tampilan HTML dengan menggunakan JSP, Java Server Faces (JSF) dan ADF Faces.

3.2.3. Arsitektur MVC

Arsitektur *Model-View-Controller* adalah sebuah pola yang terbukti membangun proyek secara lebih efektif. Hal itu dilakukan dengan memilah komponen antara *Model*, *View* dan *Controller* pada bagian-bagian dalam proyek.

1. Model

Pola MVC memiliki layer Model, merepresentasikan data yang digunakan oleh aplikasi sebagaimana proses bisnis yang diasosiasikan terhadapnya. Dengan memilahnya sebagai bagian terpisah, seperti penampungan data, *persistence*, serta proses manipulasi, terpisah dari bagian lain aplikasi.

Terdapat beberapa kelebihan dalam pendekatan ini. Pertama, membuat detail dari data dan operasinya dapat ditempatkan pada area yang ditentukan (*Model*) dibanding tersebar dalam keseluruhan lingkup aplikasi. Hal ini memberikan keuntungan dalam proses *maintenance* aplikasi.

Kedua, dengan pemisahan total antara data dengan implementasi *interface*, komponen model dapat digunakan kembali oleh aplikasi lain yang memiliki kegunaan yang hampir sama.

2. View

Layer ini mengandung keseluruhan detail dari implementasi user interface. Disini, komponen grafis menyediakan representasi proses internal aplikasi dan menuntun alur interaksi *user* terhadap aplikasi. Tidak ada *layer* lain yang berinteraksi dengan user, hanya *View*.

Penggunaan *layer View* memiliki beberapa kelebihan: Pertama, memudahkan penggabungan divisi desain dalam development team. Divisi desain dapat berkonsentrasi pada *style, look & feel*, dan sebagainya, dalam aplikasi tanpa harus memperhatikan lebih detail pada yang lain. Dan juga, memiliki *layer View* yang terpisah memungkinkan ketersediaan multiple interface dalam aplikasi. Jika inti dari aplikasi terletak pada bagian lain (Model), *multiple interfaces* dapat dibuat *(Swing, Web, Console)*, secara keseluruhan memiliki tampilan yang berbeda namun mengeksekusi komponen *Model* sesuai fungsionalitas yang diharapkan.

3. Controller

Layer ini menyediakan detail alur program dan transisi layer, dan juga bertanggung jawab pada penampungan *events* yang dibuat oleh *user* dari *View* dan melakukan *update* terhadap komponen *Model* menggunakan data yang dimasukkan oleh *user*.

Dengan menggunakan *layer* terpisah yang melakukan *update* terhadap komponen *Model*, detail tersebut dihapus dari *layer* presentasi. *Layer* presentasi

kembali pada fungsi utamanya untuk menampilkan data kepada *user*. Detail tentang bagaimana data dari *user* mengubah ketetapan aplikasi disembunyikan oleh *Controller*. Hal ini memisahkan dengan jelas antara *presentation logic* dengan *business logic*.

3.2.4. Arsitektur MVC untuk Web

Arsitektur MVC secara sederhana dirancang dan diadaPT.asi dalam penggunaan Web-Application. Arsitektur yang dihasilkan kemudian disebut dengan Model 2 *Architecture*.

Aplikasi Model 2 umumnya memiliki:

a) *Servlet Controller* yang menyediakan akses tunggal terhadap keseluruhan aplikasi, *Controller* ini bertanggung jawab menyediakan manajemen terpusat terhadap alur aplikasi dan juga *service* lain seperti penanganan *security* dan *user management*.

b) Konfigurasi XML untuk menentukan alur aplikasi dan pemrosesan perintah yang membuat *helper components* yang berfungsi sebagai *Command objects*. Hal ini berarti *helper components* terasosiasikan dengan *user actions* dan dibuat atau dipanggil untuk menangani *actions* yang terjadi, memanggil komponen *Model* yang diperlukan. Hal ini berfungsi untuk memisahkan antara *controller servlet* dan *model*.

Implementasi sebuah pola dapat dipermudah dengan menggunakan *third-party framework*. *Framework* tersebut menyediakan detail terkait (*request*, konfigurasi, dan sebagainya) sehingga kita dapat berkonsentrasi pada hal lain yang lebih penting.

3.2.5. Keuntungan Oracle ADF

1. Pengembangan Java EE secara visual dan deklaratif

Aspek yang membuat suatu *development framework* menjadi berguna adalah tersedianya *tool* pengembangan yang menyederhanakan pembuatan aplikasi dengan menggunakan *framework* tersebut. Oracle menyediakan *tool* visual dan deklaratif untuk setiap *layer* di Oracle ADF. *Tool-tool* ini terintegrasi dalam IDE Jdeveloper 11g, memberikan kemudahan bagi *developer* Java, walaupun jika mereka tidak menggunakan fitur *runtime* dari Oracle ADF.

2. Business Services Development

Oracle Jdeveloper memungkinkan berbagai macam cara dalam membangun *business services* meliputi: EJB/JPA, *web services*, Objek Java yang sederhana, ADF BC, dan lain-lain. "*Productivity with Choice*" merupakan sebuah landasan untuk pendekatan ini.

3. Pengembangan *User Interface*

Fitur pengembangan secara visual dan deklaratif di *layer View* dan *Controller* suatu aplikasi banyak tersedia di Oracle JDeveloper. Fitur-fitur tersebut antara lain:

a) Model *Page Flow* untuk ADF controller, menyediakan pengembangan model visual dari *page flow* dengan menggunakan *drag and drop* dari komponen ke diagram.
b) *Tool* pengembangan yang bersifat deklaratif untuk menambah komponen ke *user interface*, *property inspector*, komponen *palette* yang *extensible*, dan *data control pallete*.

c) Fitur *reusability,* antara lain dalam pembuatan *work flow,* ADF Libraries, dan komponen deklaratif.

d) Oracle ADF Faces, suatu *set* komponen UI untuk aplikasi *web* yang dibuat dengan menggunakan JSF, menyediakan tampilan UI yang lebih kaya.

1. ADF Business Component (ADF BC)

ADF Business Component adalah sebuah *framework* dari Oracle yang memudahkan pengembangan aplikasi bisnis dengan menggunakan platform Java EE. ADF Business Components terdiri dari:

a) *Entity Object*: *entity object* adalah sebuah komponen ADF *Business Components* yang mewakili sebuah *row* dari sebuah tabel di data *source* yang sudah ditentukan sebelumnya. *Entity object* mengenkapsulasi *business logic* untuk menjaga konsistensi *business rule*.

b) *View Object*: view object mewakili sebuah *query* SQL dan menyederhanakan langkah-langkah yang dilakukan untuk melakukan perubahan data dengan hasil dari *query* SQL.

c) *Application Module*: *application module* adalah komponen trasaksional yang digunakan UI untuk mengakses data aplikasi. *Application module* mendefinisikan data model yang *updateable* dan prosedur-prosedur dan *function-function*.

3.2.6. Java

Java adalah sebuah bahasa pemrograman yang dikembangkan oleh Sun Microsystems dan beredar di tahun 1995 sebagai bagian inti dari platform Java dari Sun Microsystems. Bahasa pemrograman Java memiliki kemiripan *sintaks*

dengan bahasa pemrograman C dan C++, tetapi memiliki model objek yang lebih sederhana dan fasilitas *low-level* yang lebih sedikit. Aplikasi-aplikasi Java akan di-*compile* dalam *bytecode* yang dapat berjalan pada Java Virtual Machine (JVM).

3.2.7. Java EE

Java Platform, Enterprise Edition (Java EE) adalah sebuah *platform* untuk server programming pada bahasa pemrograman Java. Platform Java EE berbeda dari *platform* Java Standard Edition (Java SE) dalam hal library-librarynya yang menyediakan fungsionalitas untuk men-*deploy software* Java yang terdistribusi dan multi-tier serta didasarkan pada komponen-komponen modular yang berjalan pada sebuah application server.

3.2.8. Java Server Pages (JSP)

JavaServer Pages (JSP) adalah sebuah teknologi Java yang memungkinkan para *software developer* untuk meng-*generate* HTML, XML dan dokumen-dokumen lain secara dinamis sebagai *response* dari *request client*. Dengan menggunakan JSP, *code* Java dapat disisipkan dalam *content* statis. Dalam perkembangannya format penulisan *sintaks-sintaks* JSP menjadi kompatibel dengan XML.

Keuntungan penggunaan JSP yang kompatibel dengan XML antara lain:

a) Menghindari pencampuran *code* Java dan tag-tag komponen.

b) Memudahkan *parsing* halaman untuk membuat dokumentasi.

c) Memudahkan pembuatan layer *view* yang dapat dipersonalisasi.

3.2.9. Java Server Faces (JSF)

Dengan munculnya kesadaran dari komunitas pengembang akan perlunya standarisasi *framework layer view*, Java Community Process (JCP) mulai mengembangkan JSF sebagai standar UI untuk aplikasi web yang berbasis Java. Dari rilis pertamanya di tahun 2004 hingga rilis terbarunya (JSR-252) di tahun 2006, JCP telah mengumpulkan *resources* dari komunitas, termasuk Oracle, dalam mendefinisikan spesifikasi JSF. JSF menyederhanakan pengembangan *User Interface* dengan menyediakan pendekatan yang berfokus pada komponen. Sekarang ini, JSF telah menjadi bagian dari standar Java EE.