

## **BAB II**

### **LANDASAN TEORI**

Bab ini akan akan memaparkan berbagai landasan teori yang mendukung pelaksanaan Tugas Akhir, antara lain teori mengenai antarmuka, *JavaServer™ Faces*, *ICEfaces*, *Facelets*, dan *NoteBOX*. Landasan teori ini akan memberikan pemahaman yang lebih detail mengenai topik-topik tersebut sehingga akan memudahkan proses analisis pada bab selanjutnya.

#### **2.1 Antarmuka**

Antarmuka (*user interface*) adalah bagian dari sistem komputer atau perangkat lunak yang dapat dilihat, didengar, disentuh, dan dipahami oleh pengguna [GAL02]. Antarmuka berperan penting dalam interaksi antara pengguna dan sistem perangkat lunak. Interaksi yang terjalin dengan baik akan mengoptimalkan pemakaian sistem untuk membantu pekerjaan sehari-hari. Interaksi yang dimunculkan melalui antarmuka dapat dinilai dari *usability* antarmuka tersebut.

##### **2.1.1 Usability**

*Usability* menunjukkan kemudahan pengoperasian, cara penggunaan yang mudah dipelajari dan diingat, serta kenyamanan penggunaan aplikasi. Sistem yang memiliki *usability* yang baik meminimumkan usaha fisik dan kognitif pengguna pada saat memanfaatkan sistem tersebut. Untuk menilai *usability* antarmuka sistem, para pakar *usability* mengajukan kriteria pengukuran seperti yang tertera pada Tabel II-1.

Secara umum kriteria nomor 1 sampai dengan 8 pada Tabel II-1 dikategorikan sebagai perfomansi pengguna (pengukuran yang bersifat obyektif) dan kriteria nomor 9 dan 10 dikategorikan sebagai pengukuran yang sifatnya subjektif dari sudut pandang pengguna. Berikut adalah uraian kriteria berdasarkan empat referensi pada Tabel II-1 tersebut.

1. Efektivitas

Shackel mendefinisikan efektivitas sebagai perfomansi pengguna memakai sistem yang bersangkutan [SHA90]. Perfomansi tersebut diukur dari waktu yang diperlukan pengguna untuk menyelesaikan sebuah tugas (*task*) dan dari

jumlah kesalahan yang dibuat. Standar ISO memakai efektivitas sebagai indikator keakurasian dan kelengkapan sistem memenuhi kebutuhan pengguna. Sementara efektivitas menurut Preece mengacu pada kualitas sistem dalam penyelesaian tugas-tugas sesuai dengan tujuan pembuatan sistem tersebut.

**Tabel II-1 Kriteria Pengukuran Usability**

No.	Kriteria	Shackel (1990)	Nielsen (1993)	ISO 9241-11 (1998)	Preece dkk. (2002)
1.	Efektivitas	√		√	√
2.	Efisiensi		√	√	√
3.	<i>Learnability</i>	√	√		
4.	<i>Memorability</i>	√	√		
5.	Fleksibilitas	√			
6.	Kesalahan		√		
7.	Utilitas				√
8.	<i>Safety</i> (Keamanan)				√
9.	Kepuasan		√	√	
10.	<i>Attitude</i> (Perilaku)	√			

## 2. Efisiensi

Nielsen mengaitkan efisiensi dengan kebutuhan sumber daya, seperti usaha, waktu, dan biaya, untuk mencapai tujuan pemakaian sistem tersebut [NIE93].

Standar ISO mengaitkan efisiensi dengan hubungan antara sumber daya yang dibutuhkan dan tujuan yang tercapai – semakin efisien sebuah sistem semakin cepat pengguna mendapatkan tujuannya.

Preece mengukur efisiensi dari optimasi sistem mendukung pengguna dalam penyelesaian tugas-tugasnya sesuai kemampuan sistem tersebut [PRE02].

## 3. *Learnability*

Shackel dan Nielsen berpendapat *learnability* sebagai tingkat kemudahan sistem untuk dipelajari, diukur melalui waktu yang diperlukan untuk mempelajari penggunaan sistem hingga mencapai level kemahiran tertentu [SHA90][NIE93].

## 4. *Memorability*

Berbeda dengan Shackel yang memasukkan unsur *memorability* sebagai bagian dari *learnability*, Nielsen berpendapat bahwa kriteria *memorability*

berdiri sendiri terlepas dari kriteria *learnability*. *Memorability* berhubungan dengan proses *recalling* (mengingat) cara pemakaian sistem setelah pengguna tidak berinteraksi dengan sistem tersebut selama beberapa waktu [NIE93].

5. *Fleksibilitas*

Shackel memandang perlunya sebuah sistem memiliki atribut fleksibilitas. Fleksibilitas berkaitan dengan variasi pengerjaan suatu *task*/fungsi sistem [SHA90]. Pengguna dapat mengakses lebih dari satu cara untuk melakukan suatu *task*.

6. *Kesalahan*

Nielsen menambahkan kriteria kesalahan dalam menilai *usability* sebuah sistem. Frekuensi kesalahan yang tinggi pada saat penggunaan sistem mengindikasikan rendahnya *usability* sistem yang bersangkutan.

7. *Utilitas*

Preece dkk. memakai utilitas sebagai acuan tingkat fungsionalitas sebuah sistem yang dapat digunakan pengguna untuk menyelesaikan suatu tugas. Contohnya, perangkat lunak akuntansi yang menyediakan fitur perhitungan pajak memiliki utilitas tinggi, sedangkan perangkat lunak grafik yang memaksa penggunanya menggunakan *mouse* untuk menggambar, memiliki utilitas rendah.

8. *Safety* (Keamanan)

Preece dkk juga mempertimbangkan masalah *safety* (keamanan) sebagai sebuah kriteria *usability*. Sistem keamanan mencegah kerusakan fatal pada sistem dari kondisi yang tidak diinginkan. Tidak hanya itu, sistem juga memberikan petunjuk perbaikan apabila terjadi kesalahan.

9. *Satisfaction* (Kepuasan)

Kriteria kepuasan menjadi pertimbangan bagi Nielsen dan standar ISO. Kepuasan pengguna terhadap sistem yang dipakainya mengindikasikan bahwa sistem tersebut layak pakai.

10. *Attitude* (Perilaku)

Shackel mengukur kriteria perilaku sistem dari bagaimana pengguna menerima dan merasa puas dari sistem yang dipakainya [SHA90].

### 2.1.2 Antarmuka pada Aplikasi Web

Antarmuka web terdiri dari dua komponen utama yaitu navigasi dan informasi [GAL02]. Pengguna berinteraksi dengan isi web melalui navigasi dan bagaimana web tersebut menampilkan informasi yang diminta pengguna. Oleh karena itu, perancangan antarmuka web difokuskan pada navigasi dan tampilan informasi. Antarmuka web yang baik menyatukan semua elemen web yang terdiri dari menu, isi, dan tautan dokumen atau grafik, secara natural, terstruktur, dan mudah diakses pengguna.

Navigasi adalah salah satu komponen penting pada antarmuka web. Pada sebuah situs web, navigasi berfungsi sebagai penunjuk arah sehingga pengunjung tahu apa yang harus dilakukan untuk mengakses informasi yang diinginkan. Selain berfungsi sebagai penunjuk arah, navigasi sangat berperan menentukan kesuksesan situs web.

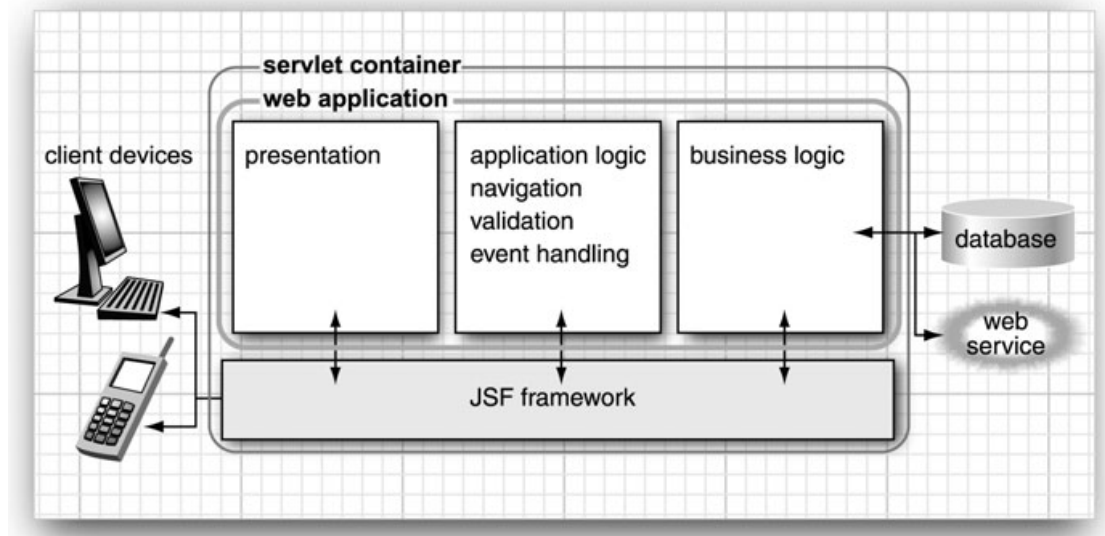
Untuk merancang navigasi web yang baik, ada beberapa panduan yang dikemukakan oleh IBM [IBM05], yaitu:

1. Menggunakan keterangan yang mengindikasikan fungsi navigasi,
2. Memberikan informasi kepada pengunjung mengenai letak posisinya dalam struktur web yang sedang diakses,
3. Menggunakan elemen navigasi secara konsisten,
4. Menyediakan taut absolut ke halaman awal web di tiap halaman web tersebut,
5. Memastikan area grafik dapat diakses oleh pengguna yang memiliki cacat penglihatan.

## 2.2 *JavaServer Faces*

*JavaServer Faces* atau biasa disingkat JSF adalah *user interface framework* dalam bahasa Java untuk membangun aplikasi web [JEN07]. JSF merupakan salah satu bagian dari teknologi yang membangun *platform* Java EE. JSF diciptakan pada tahun 2002 melalui *Java Specification Request* (JSR) 127. JSR 127 kemudian mengalami banyak perubahan dan pada bulan Maret 2004 diluncurkan versi finalnya [MAN05]. Salah satu kelebihan utama dari JSF adalah teknologi ini menawarkan pembagian yang jelas antara layer presentasi dan bisnis.

### 2.2.1 Arsitektur *JavaServer Faces*



Gambar II-1 Arsitektur *JavaServer Faces*

Pada gambar II-1 terlihat bahwa JSF bertanggung jawab dalam menangani interaksi klien dan aplikasi, menghubungkan bagian presentasi, logik aplikasi, dan bisnis logik menjadi suatu aplikasi web. Subsistem lainnya, seperti layanan EJB atau basis data dapat diintegrasikan dengan mudah walaupun bukan merupakan bagian dari JSF.

JSF mengikuti arsitektur *Model-View-Controller* (MVC) dengan pembagian sebagai berikut:

1. *Model* adalah logik bisnis dan data yang bisa berupa EJB, basis data, atau yang lainnya.
2. *View* adalah layer presentasi yang berinteraksi langsung dengan pengguna. *View* bisa berupa JSP, atau teknologi *display* lainnya.
3. *Controller* adalah kode aplikasi yang menangani *events* dan menghubungkan *model* dan *view*. Dalam JSF, *servlet* berperan sebagai *controller*.

### 2.2.2 Elemen Pembentuk *JavaServer Faces*

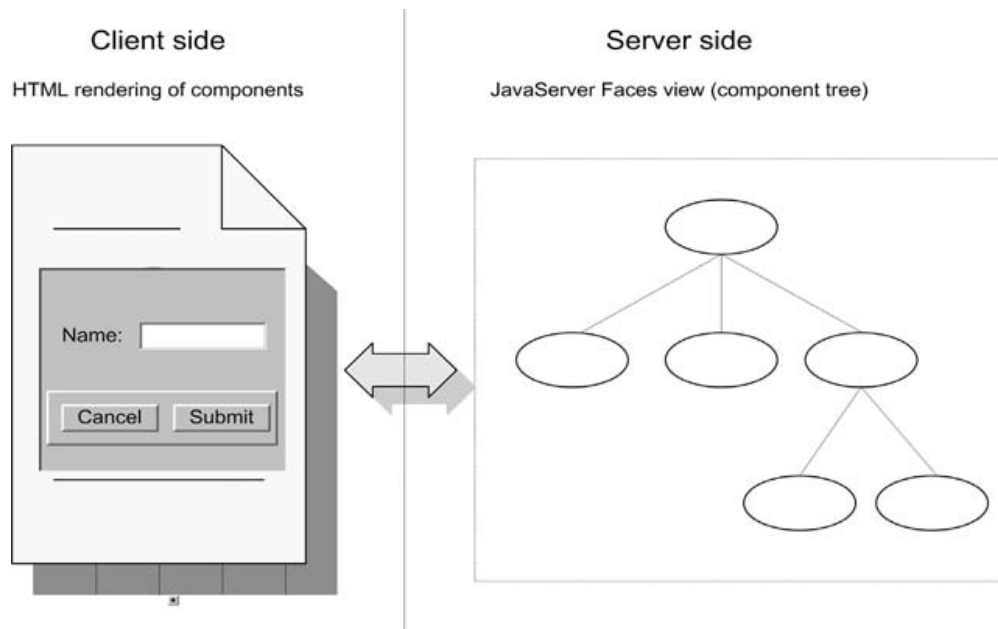
Seperti *framework* lainnya, JSF mempunyai elemen-elemen penyusun yang menyediakan rangkaian fitur yang dapat digunakan oleh pengembang aplikasi web. Pada dasarnya, terdapat delapan elemen utama yang perlu diketahui, yaitu:

#### 1. **Komponen *User Interface* (UI).**

Komponen UI adalah objek yang menangani interaksi dengan *end user*. Komponen tersebut mirip dengan kontrol pada Swing yang dibangun di atas

*JavaBeans*. Walaupun sama-sama memiliki properti, *method*, dan *events*, komponen UI didesain dengan batasan yang unik untuk aplikasi web, dan hidup di server, bukan di klien.

Komponen pada JSF dapat mengingat nilai antara *request* yang satu dengan yang lain dikarenakan JSF menyimpan representasi pohon keterhubungan dari komponen-komponen pada halaman web. Pohon komponen ini disebut *view* dan merupakan representasi internal dari JSF terhadap suatu halaman web. Pohon itu menggambarkan hubungan bapak-anak dari komponen. Sebagai contoh, sebuah *form* mengandung label, *text field*, dan *panel* dengan dua tombol seperti pada gambar II-2. Dalam hal ini, komponen *form* merupakan bapak dari komponen label, *text field*, dan *panel*.



**Gambar II-2 Komponen UI dikelola oleh server dalam bentuk pohon**

Semua komponen pada JSF merupakan turunan dari kelas `UIComponentBase` yang mendefinisikan status *default* dan perilaku dari komponen tersebut. Beberapa komponen baku telah disediakan oleh JSF antara lain *textbox*, *panel*, *label*, *data grid*, *graphic*, *listbox*, *radio button*, *checkbox*, dsb. Selain komponen-komponen baku tersebut, masih ada komponen lainnya seperti *toolbar*, *menu*, *file upload*, *tabbed panel*, dsb.

Komponen dapat dideklarasikan pada halaman *markup* sebagai berikut:

```
<h:inputText id="helloInput" value="default" required="true">
```

Komponen tersebut dapat dimanipulasi pada kode Java di server dengan cara sebagai berikut:

```
...
HtmlInputText input = (HtmlInputText)event.getSender();
input.setDisabled(true);
input.setStyle("color: blue");
...
```

## 2. *Renderer.*

*Renderer* bertanggung jawab dalam melakukan *encoding* dan *decoding* komponen. *Encoding* adalah proses menampilkan komponen pada halaman web sehingga dapat dimengerti dan dilihat oleh pengguna, sedangkan *decoding* mengekstraksi parameter *request* yang benar dan kemudian mengeset *value* dari komponen berdasarkan parameter tersebut. Sebagai contoh, kode berikut mendefinisikan sebuah komponen `HtmlInputText`.

```
<h:inputText id="inputText" size="20" maxLength="30"/>
```

Ketika di-*encode*, kode tersebut akan diubah menjadi kode HTML yang sesuai dengan teknologi *view* yang digunakan oleh pengguna.

```
<input id="myForm:inputText" type="text"
name="myForm:inputText" maxLength="30" size="20" />
```

Ketika pengguna memasukkan teks "foo" ke dalam komponen ini, maka proses *decoding* akan mengambil parameter *form* yang dikirim melalui HTTP *response* dan mengeset nilai dari instans komponen `HtmlInputText` di server ke "foo".

*Renderer* diorganisir ke dalam *render kits* yang umumnya dibedakan berdasarkan tipe keluarannya. JSF telah menyertakan *render kit* standar untuk HTML 4.01.

## 3. *Backing beans.*

*Backing beans* merupakan objek yang menangani interaksi antara *view* dan *model* pada arsitektur MVC. *Backing beans* umumnya mengandung properti atau masukan dari user dan juga *event listener methods* yang memproses properti tersebut.

JSF memungkinkan suatu *backing beans* memiliki asosiasi dengan komponen UI menggunakan JSF *expression language* (EL). JSF EL cukup serupa dengan JSP 2.0 ataupun JSTL *expression language*. *Programmer* dapat

menghubungkan suatu properti *backing beans* dengan *value* dari suatu komponen seperti pada contoh di bawah.

```
<h:outputText id="helloBeanOutput"
value="#{helloBean.numControls}"/>
```

Kode di atas menunjukkan bahwa *programmer* mengasosiasikan (*bind*) properti *numControls* dari objek *helloBean* ke nilai komponen *outputText*. Apabila nilai pada komponen berubah, maka properti dari objek yang berasosiasi pun ikut berubah.

#### 4. *Validators*.

*Validators* melakukan verifikasi terhadap nilai suatu komponen. Sebuah komponen UI dapat berasosiasi dengan satu atau banyak jenis *validator*. JSF sendiri menangani validasi dengan tiga cara, yaitu pada level komponen UI, level *backing beans* melalui *validator method*, atau lewat kelas validator. Komponen UI biasanya hanya menangani validasi yang sederhana seperti mengecek suatu nilai harus ada atau tidak. *Validator method* berguna apabila dibutuhkan validasi pada satu atau banyak *field* dalam suatu *form*. *External validator* berguna untuk kasus yang generik misalnya panjang maksimal masukan atau kisaran angka masukan. *Validator* jenis ini bersifat *pluggable*, dapat diasosiasikan dengan banyak komponen.

JSF menyertakan berbagai macam validator standar untuk mengecek kisaran atau panjang. Ketika ditemukan kesalahan, misalnya *string* yang terlalu panjang, maka *validator* akan menambahkan *error message* pada daftar *message*. Ini akan sangat memudahkan apabila pesan *error* ingin ditampilkan.

```
<h:inputText>
    <f:validateLength minimum="2" maximum="10" />
</h:inputText>
```

Kode di atas memperlihatkan suatu komponen *HtmlInputText* yang berasosiasi dengan *Length validator* yang mengecek masukan dari pengguna agar panjangnya kurang dari dua dan tidak lebih dari sepuluh karakter.

#### 5. *Converters*.

Ketika pengguna berinteraksi dengan aplikasi JSF, maka mereka sebenarnya berinteraksi dengan output dari *renderers*. Untuk melakukan ini, *renderers* harus mempunyai pengetahuan mengenai komponen yang ditampilkan. Hal ini menjadi masalah karena komponen dapat berasosiasi dengan properti *backing*

*beans* yang dapat memiliki beragam tipe, seperti *String*, *Date*, atau tipe bentukan lainnya. Tidak adanya batasan dari tipe-tipe tersebut menyebabkan *renderer* tidak dapat mengetahui bagaimana cara menampilkan objek tersebut. *Converters* hadir untuk menangani masalah ini. *Converters* bertanggung jawab dalam mengubah nilai dari komponen ke bentuk *String* untuk ditampilkan, dan dari masukan *String* kembali ke objek semula. Selain itu, *converters* juga dapat menangani *formatting* dan *localization*. Sebagai contoh, *DateTime converter* dapat melakukan format pada objek *Date* dalam bentuk *short*, *medium*, *long*, atau *full style*. Untuk setiap bentuk, *converters* akan menampilkannya sesuai dengan *user's locale*.

Berikut cara untuk me-*register* suatu *converter* ke dalam komponen.

```
<h:outputText value="#{user.dateOfBirth}">
    <f:convert_datetime type="both" dateStyle="full"/>
</h:outputText>
```

Misalkan tanggal lahir dari pengguna adalah 4 Mei 1942, maka komponen *HtmlOutputText* akan menampilkan *string* "05/04/42" apabila pengguna berasal dari Inggris atau menampilkan "04/05/42" jika berasal dari Canada.

## 6. *Events dan listeners.*

*Event* merepresentasikan interaksi yang pengguna lakukan terhadap komponen UI. Suatu *event* dapat berupa memencet/mengklik suatu komponen, menyorot komponen, atau mengeksekusi suatu perintah yang spesifik. JSF menggunakan model *JavaBeans events* seperti *Swing*. Suatu objek dapat menciptakan banyak *events* yang kemudian akan ditangkap oleh *listeners*. *Listener* dapat diimplementasikan pada *backing beans method* atau kelas *listener* sendiri.

Contoh:

```
<h:commandButton type="submit" value="Login"
action="#{loginForm.login}"/>
```

Kode di atas menunjukkan suatu komponen *CommandButton* yang me-*register action method* "login". Ketika tombol diklik, maka *action event* akan dipanggil, dan fungsi "login" akan dieksekusi.

## 7. *Messages.*

JSF mempunyai *built-in support* untuk menampilkan pesan pada aplikasi. Pesan pada JSF dapat dihasilkan baik oleh *validator*, *converter*, komponen UI,

*kode aplikasi*, ataupun *event listener*. Selain itu, pesan dapat diasosiasikan dengan komponen pada halaman web.

Contoh:

```
<h:message id="errors" for="helloInput" style="color: red"/>
```

Tag pesan ini akan menampilkan semua error yang dihasilkan oleh komponen dengan id “*helloInput*”.

## 8. *Navigation*.

*Navigation* adalah proses perpindahan dari salah satu halaman web ke halaman lainnya. JSF mempunyai sistem navigasi yang elegan yang bersifat deklaratif. *Programmer* dapat mendefinisikan perpindahan ke suatu halaman melalui keluaran yang dihasilkan oleh suatu *action method*.

```
<navigation-rule>
  <from-view-id>/login.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/mainmenu.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-outcome>
    <to-view-id>/login.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Contoh di atas memperlihatkan dua kasus pada halaman *login.jsp*, yaitu untuk keluaran “*success*” dan “*failure*”. Untuk setiap halaman yang diberikan, *navigation rule* mendefinisikan keluaran dan halaman yang dipanggil sesuai dengan keluaran tersebut. Untuk satu hubungan antara keluaran dan halamannya didefinisikan dalam *navigation case*. Jika *action method* pada *login.jsp* menghasilkan *string* “*success*” maka navigasi akan memanggil halaman *mainmenu.jsp*, dan sebaliknya jika “*failure*” maka navigasi tetap di halaman *login.jsp*.

## 2.3 *ICEfaces*

*ICEfaces* merupakan pengembangan dari *framework* JSF yang dilengkapi dengan fitur interaktif berbasis AJAX [ICE07]. *ICEfaces* mengganti *renderers* standard dari JSF yang berbasis HTML dengan *Direct-to-DOM* (D2D) *renderers*, dan memperkenalkan

*lightweight AJAX bridge* untuk melakukan perubahan pada layer presentasi dan mengirimkan *events* ke server. Selain itu, *ICEfaces* juga menyediakan serangkaian *AJAX-enabled component* untuk membangun aplikasi web yang interaktif.

Beberapa teknologi kunci yang diimplementasikan oleh *ICEfaces* adalah sebagai berikut:

**1. *Direct-to-DOM (D2D) Rendering.***

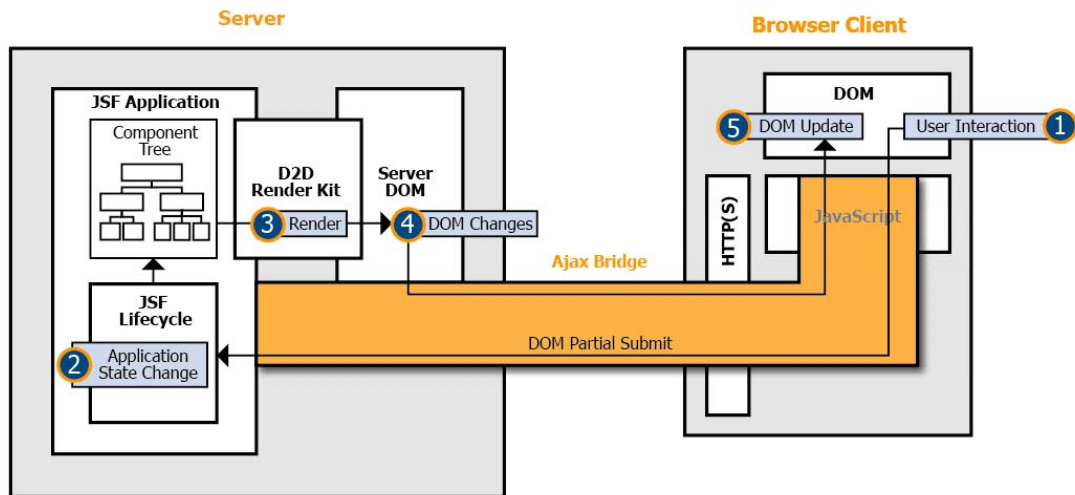
Dengan teknik ini, *render cycle* dari JSF dapat melakukan *render* terhadap pohon komponen JSF langsung ke dalam suatu DOM yang disimpan di server (*server-resident DOM*). *ICEfaces* telah menyediakan kumpulan komponen yang memanfaatkan teknologi D2D ini. DOM (*Document Object Model*) adalah model objek untuk merepresentasikan HTML atau XML dan format lain yang berkaitan. DOM dibutuhkan oleh AJAX untuk memeriksa atau memodifikasi halaman web secara dinamis. Dengan kata lain, DOM adalah cara AJAX atau *Javascript* melihat halaman HTML dan *state browser*.

**2. *Incremental Browser DOM Update.***

Pada *ICEfaces*, hasil dari D2D *rendering* adalah suatu DOM yang merepresentasikan *state* baru dari layer presentasi. Dengan mengirimkan DOM baru ini ke *browser* pengguna secara bertahap untuk menggantikan DOM yang lama, maka perubahan pada halaman dapat terjadi hampir tanpa adanya proses *rendering* ulang. Hal ini dimungkinkan dengan adanya *lightweight AJAX bridge* yang mengkomunikasi perubahan DOM secara bertahap dan menyusun kembali perubahan tersebut pada DOM di *browser*.

**3. *Automated Partial Submit.***

Komponen *ICEfaces* memiliki atribut yang dapat men-*submit* nilainya ke server secara parsial (*partial submit*). Hal ini memungkinkan pengembang untuk memodifikasi komponen agar dapat melakukan *partial submit* secara otomatis berdasarkan interaksi pengguna dengan komponen tersebut.



**Gambar II-3 Incremental Page Update dengan D2D Rendering**

Gambar II-3 menggambarkan proses *update* halaman secara bertahap (*incremental page update*) menggunakan *D2D rendering* [MAR06]. Langkah-langkahnya adalah sebagai berikut:

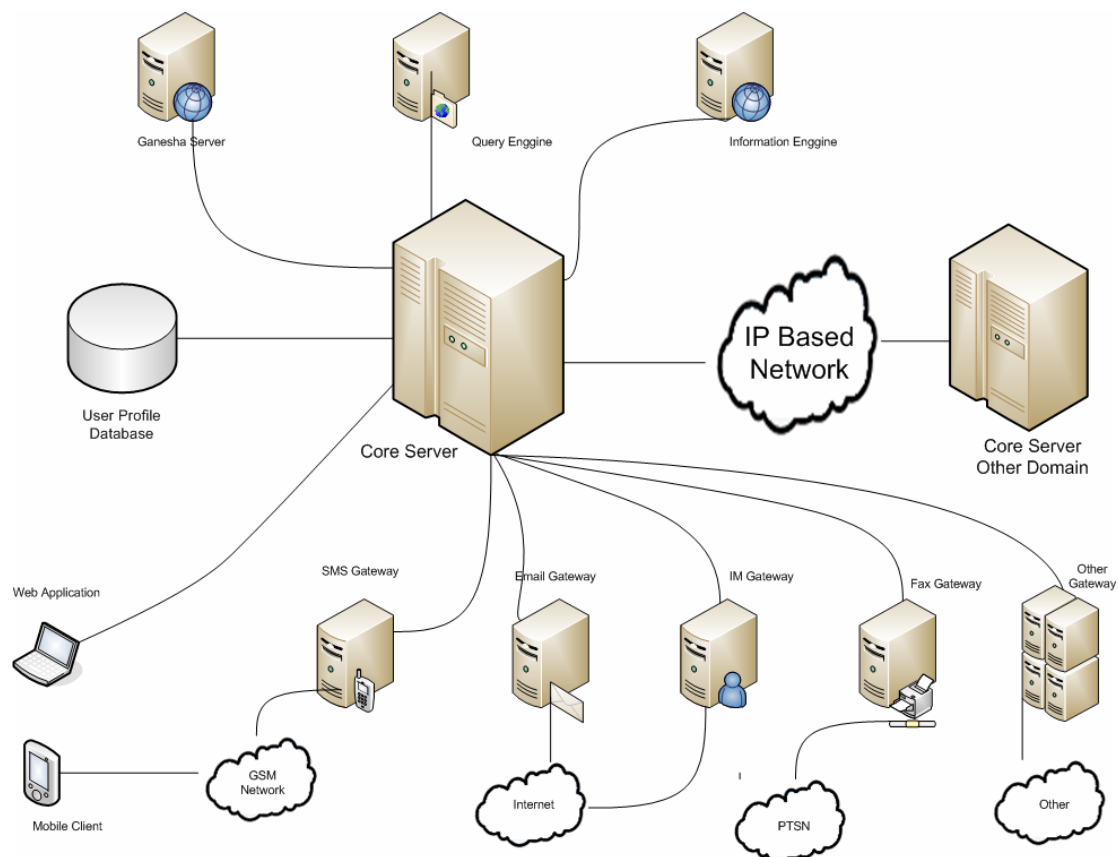
1. Interaksi pengguna dengan halaman pada *browser* menyebabkan *automated partial submit* melalui *bridge*.
2. Siklus hidup aplikasi JSF dijalankan. Logik bisnis aplikasi merespon interaksi pengguna tadi dan menghasilkan perubahan *state* pada aplikasi.
3. Proses *render* D2D dijalankan dan kemudian menghasilkan DOM baru (server DOM).
4. Perubahan DOM dikirim secara bertahap ke *browser* melalui *AJAX bridge*.
5. Penyusunan ulang DOM pada *browser*.

## 2.4 Facelets

*Facelets* adalah implementasi *open source* yang baru muncul ke permukaan yang menyediakan *template engine* untuk JSF dan mengeliminasi *JSP processing*. *Facelets* menggantikan JSP pada layer presentasi JSF. Pengembangan JSF yang menggunakan JSP sebagai *view renderer* akan banyak memunculkan *taglib* yang bukan merupakan *tag* HTML. Dengan *facelets*, pengembang dapat memasukkan komponen ke dalam elemen HTML biasa.

## 2.5 NoteBOX

NoteBOX adalah suatu sistem *mobile unified messaging* yang bersifat *open source*. *Mobile unified messaging* adalah sebuah sistem yang memungkinkan penggunaan berbagai macam layanan *messaging* dalam sebuah sistem tunggal pada jaringan *mobile* berbasis *Internet Protocol (IP)* [CAN06]. NoteBOX dibangun untuk menyediakan layanan *messaging* dan memberikan kemudahan pada masyarakat dalam mengakses konten berupa informasi dan pengetahuan umum. Sebagai contoh adalah pengguna dapat melakukan pencarian informasi mengenai suatu hal, misalnya saja pencarian informasi mengenai kayu. Dengan mengirimkan pesan pada NoteBOX maka pengguna akan mendapatkan beberapa informasi mengenai kayu, misalnya dari jenis-jenis kayu yang ada sampai pada dimana dapat memperoleh kayu tersebut (informasi yang dikirimkan pada pengguna tergantung pada informasi yang terdapat pada *content server*).



**Gambar II-4** Arsitektur Global Sistem NoteBOX

Seperti terlihat pada gambar II-4, NoteBOX adalah sebuah sistem yang tersusun atas tiga bagian/layer. Bagian tersebut adalah *user layer* NoteBOX, *core layer* NoteBOX,

dan *application layer* NoteBOX. Setiap bagian tersebut menjalankan fungsinya masing-masing. *User layer* menangani masalah-masalah ataupun proses-proses yang berhubungan langsung dengan *device* yang dipakai *user*. *Core layer* merupakan jantung sistem NoteBOX. Di dalamnya terdapat *core server* yang bertugas mengatur lalu lintas komunikasi antara sesama *gateway* pada *user layer* ataupun komunikasi antara *user layer* dan *application layer*. Selain itu *core server* juga bertanggung jawab untuk berkomunikasi dengan mesin lain yang berada di luar NoteBOX. Bagian akhir yaitu *application layer* bertanggung jawab untuk memberikan layanan sesuai dengan permintaan pengguna. Setiap bagian dalam noteBOX dibuat untuk mendukung penggunaan prinsip *multiple messaging*.

### 2.5.1 Multiple Messaging

*Multiple-messaging* adalah prinsip yang diterapkan dalam NoteBOX dimana pengguna dari dapat memilih kemana/dalam bentuk apa balasan yang hendak diterimanya. Misalnya saja pengguna ingin menerima balasan dalam ponselnya (dalam bentuk SMS atau MMS) setelah ia melakukan permintaan melalui terminal komputer yang dipakainya (melalui *email*).

Dengan prinsip tersebut maka pengguna akan mendapatkan kebebasan dalam menggunakan fasilitas ini dan noteBOX ini dapat digunakan oleh banyak orang karena kemampuannya dalam melayani berbagai jenis *device* yang dipakai penggunanya.

### 2.5.2 User Layer

Bagian *user layer* dari NoteBOX adalah lapisan tempat *user* dan *client* dari sistem berada. *Client* dapat berupa *native client*, yaitu aplikasi *client* yang dijalankan di terminal milik *end user* (*User Environment*), dan *gateway*, program yang menjembatani antara NoteBOX dengan sistem eksternal seperti sistem SMS, *email*, faksimil, dan sebagainya. *Native client* dapat berupa aplikasi pada *handset* atau aplikasi berbasis web yang dapat diakses *user*. *Gateway* adalah program yang bertanggung jawab mengubah pesan atau data yang diterima dari *user* ke dalam bentuk yang dikenali dan meneruskan ke layer di atasnya (*core layer*) untuk diproses. *Gateway* yang saat ini tersedia adalah *SMS gateway*, dan *email gateway*. Selain melakukan konversi bentuk atau format dari *device user* maka *layer* ini juga melakukan fungsi sebaliknya yaitu mengubah format pesan atau data dari format yang

dikenali NoteBOX ke dalam format yang dapat diakses atau digunakan oleh *device* penerima.

Perkembangan kedepannya adalah semakin banyak format pesan atau data yang dapat dikenali sehingga membuat komunikasi menjadi semakin mudah. Hal tersebut dikarenakan orang tidak memerlukan suatu *device* khusus (misal mesin faksimil) untuk menerima suatu pesan atau data karena pesan atau data tersebut dapat diterima melalui *device* yang dimiliki (semisal *email* pada computer pribadi).

### **2.5.3 Core Layer**

*Core layer* merupakan lapisan yang menjalankan fungsi-fungsi utama dari sistem NoteBOX. Bagian ini bertanggung jawab terhadap lalu lintas data antara *user layer* dan *application layer*. Pada *layer* ini terdapat suatu bagian yang dinamakan *core server*. *Core server* mengenali permintaan yang masuk yang berasal dari *user layer* dan meneruskan permintaan tersebut kepada mesin yang ada pada *application layer* sesuai dengan jenis permintaannya. Misalkan permintaan yang diterima adalah permintaan untuk mencari *email* yang diinginkan pengguna maka permintaan tersebut akan diteruskan oleh *core server* kepada mesin pengolah *email* yang ada pada *application layer*. *Core server* juga mengenali *account* dari pengguna yang melakukan permintaan. Jika *account* pengguna yang melakukan permintaan layanan tidak bermasalah maka permintaan dapat diteruskan jika terdapat suatu masalah maka *core server* ini akan memberikan informasi mengenai masalah yang dimiliki pengguna tersebut.

Selain meneruskan permintaan dari *user layer* ke *application layer*, *core server* juga bertanggung jawab dalam pengiriman hasil permintaan yang berasal dari *user layer* kepada *application layer*. Hasil permintaan tersebut akan diteruskan pada *gateway* yang sesuai dengan keinginan pengguna saat melakukan permintaan tersebut atau berdasarkan pilihan *default* yang sebelumnya telah dimasukkan oleh pengguna.

*Core server* juga memiliki kelebihan untuk berkoordinasi atau bekerja sama dengan sistem pada domain lain. Untuk itu, *Core server* akan mengatur lalu lintas data dari salah satu bagian dari NoteBOX menuju sistem atau mesin lainnya yang berada di

luar dari sistem NoteBOX ini dan juga arah sebaliknya yaitu ketika ada mesin atau sistem lain yang ingin menggunakan layanan dari salah satu bagian dari NoteBOX.

#### **2.5.4 Application Layer**

Bagian ini merupakan bagian terakhir yang ada dalam arsitektur NoteBOX. Pada bagian ini semua permintaan yang ada dieksekusi untuk mendapatkan hasil yang diinginkan. Pada bagian ini terdapat beberapa mesin yang ada, yaitu *content server* dan *search engine*. Setiap mesin tersebut bertanggung jawab terhadap permintaan yang datang dan akan memberikan hasil sesuai dengan jenis permintaan pengguna.

Pengembangan pada bagian ini sangat diperlukan karena dengan banyaknya variasi dari layanan yang ada dalam bagian ini maka akan semakin banyak layanan yang dapat ditawarkan oleh NoteBOX .

Layer ini menyediakan layanan tambahan yang diberikan oleh NoteBOX. Layanan utama yang berupa *unified messaging* dapat digunakan tanpa layanan dari layer ini. Hal tersebut dikarenakan pesan atau data yang masuk langsung diteruskan pada *gateway device* yang diinginkan, jika pengguna hanya menginginkan untuk mengirim pesan atau data. Tetapi jika pengguna ingin mencari informasi mengenai suatu hal atau ingin mendapatkan beberapa data yang dicari maka layer ini akan digunakan.

#### **2.5.5 Content Server**

*Content server* adalah *server* yang bertanggung jawab terhadap permintaan berupa *content* atau isi berdasarkan permintaan pengguna. *Server* ini memberikan layanan berupa pemberian informasi mengenai data yang dicari (berdasarkan *keyword* tertentu) dalam bentuk seperti suatu ensiklopedia. Pengembangan lebih lanjut mengenai *content* apa yang dapat diimplementasikan dalam masih terus dalam pengembangan.

Data yang terdapat dalam *server* ini dimasukan secara *manual* oleh beberapa orang yang bertugas untuk menambahkan informasi ke dalam *server* tersebut.

### **2.5.6 Search Engine**

Dalam *application layer* terdapat juga *server* yang menyediakan layanan pencarian terhadap masukan atau *keyword* yang dikirimkan pengguna. Pencarian akan dilakukan seperti mesin pencari yang terdapat dalam internet. Hasil pencarian dikirimkan pada *device* yang diinginkan pengguna.

Contoh dari layanan pencarian yang dapat dilakukan adalah pencarian mengenai lokasi. Misalkan pengguna mencari lokasi suatu toko buku dalam daerah tertentu maka dari *keyword* yang diberikan pengguna tersebut, *server (search engine server)* akan mencarinya dan jika ditemukan akan memberikan hasil pencarian berupa lokasi beberapa toko buku yang terdapat pada daerah tersebut atau informasi bahwa tidak ditemukan jika hasil pencarian nihil.