INFOMATEK
Volume 18 Nomor 2 Desember 2016

STUDI POLA PENGEMBANGAN WEB DINAMIS DENGAN ARSITEKTUR MVC: IMPLEMENTASI PADA JAVASERVER FACES (JSF)

Hendra Komara[*)]

Program Studi Teknik Informatika
Fakultas Teknik – Universitas Pasundan

Abstrak: Perangkat lunak berbasis *web* dinamis (PLBWD) melakukan pengubahan informasi melalui pengubahan data, bukan melalui pengubahan kode program. Teknologi pengembang PLBWD terus merilis dan memutakirkan kakas pengembang PLBWD dengan keunggulan masing-masing. Dari sisi arsitektur, desaian MVC menawarkan PLBWD yang memisahkan tanggung jawab dari tiap lapisan aplikasi dan kegunaulangan. Banyaknya *platform* untuk mengembangkan PLBWD, melahirkan kompleksitas dan biaya besar pada peringkat teknis. Para pemrogram memerlukan usaha besar untuk mengenali dan memahami teknologi, arsitektur, teknik, perilaku, lingkungan dari tiap *platform*. Pada penelitian ini dikembangkan pola umum PLBWD untuk meredam kompleksitas dari detail tiap *platform,* sehingga dapat meningkatkan produktivitas pemrogram. Teknik analisis pola PLBWD pada tiap *platform* dilakukan dengan cara eksperimen pada beragam aplikasi yang sudah jadi. Hasil analisis direkam, dievaluasi secara iteratif, sehingga pola PLBWD akan terus berevolusi. Pada penelitian ini teknologi dari masing-masing *platform* yang diteliti : **Java EE** : JSF; **.NET** : .NET Framework; **PHP** : YII. Untuk menguji pola PLBWD, dikembangkan sebuah kakas generator. Kakas tersebut dapat membangkitkan kode program dengan mengimplementasikan pola PLBWD yang sudah dibuat. Pada penelitian ini, implementasi hanya dilakukan pada *platform* Java EE dengan mengunakan teknologi JSF.

Kata kunci: PLBWD, pola, *platform*, MVC.

I. PENDAHULUAN

1.1 Latar Belakang

Penggunaan *web* dinamis yang meluas dan kebutuhan yang meningkat melahirkan kompleksitas dari *web* dinamis itu sendiri. Diperlukan teknik dan teknologi untuk dapat membangun aplikasi *web* dinamis yang dapat mempermudah dalam penyusunan *user interface* (UI) kompleks, adaptif terhadap perubahan, kemudahan dalam perawatan dan pengembangan (Burns [1]).

Untuk meredam kompleksitas perkembangan dan perubahan, maka rancangan aplikasi harus bersifat modular. Aplikasi modular mampu mengurangi *coupling* diantara komponen [1]. Komponen adalah entitas yang memilki fungsi tertentu dan dapat diguna ulang (*reuseable*) (Caytiles [2]). Pengaksesan komponen hanya melalui *interface*, sedangkan detail dari komponen itu sendiri tersembunyi (*encapsulation*) [1].

[*)] hendra.komara@unpas.ac.id

Salah satu kendala pembangunan perangkat lunak yang mendasar adalah tuntutan menghasilkan produk yang berkualitas dengan batas waktu pengerjaan singkat (Jiang [3]). Dari sisi pemrogram, ketika harus mempelajari teknologi baru dari sebuah *platform* secara rinci dan teknis, memerlukan usaha besar, hal tersebut dapat menghambat produktivitas pemrogram yang berdampak pada pengembangan aplikasi secara kesuluruhan.

Penelitian ini bertujuan untuk menghasilkan pola pengembangan dari tiga *platform* pengembang PLBWD*,* yaitu Java EE, .NET, PHP. Dari tiap *platform* dipilih satu teknologi untuk diteliti, yaitu 1) : PHP dipilih teknologi YII. Alasanya adalah karena YII membantu pengembang membangun aplikasi *web* kompleks secara cepat sehingga *delivery* pada pelanggan dapat tepat waktu; 2) Java EE dipilih JSF. Alasanya adalah JSF menawarkan solusi pembangunan *web* dinamis dengan kombinasi arsitektur desain MVC yang *powerfull* dan penyusunan UI yang berbasis komponen terstandar Java EE [1]; 3) .NET dipilih .NET Framework. Alasan pemilihan .NET karena semua teknologi yang dikembangkan Microsoft meski berbayar relatif matang dan memiliki kemudahan dalam penggunaan serta memiliki dukungan penuh pada hal *help desk*.

Pola yang dihasilkan mampu adaptif untuk semua *platform*. Dihasilkannya pola yang adaptif pada berbagai *platform* dapat meningkatkan produktivitas pemrogram, meski harus berganti *platform* dan teknologi dalam pengembangan PLBWD.

Untuk memvalidasi pola yang dibuat, dibuat aplikasi generator untuk membangkitkan kode program, dimana desain aplikasi yang dihasilkan oleh kode generator mengacu pada pola PLBWD yang dikembangkan.

1.2 Identifikasi Masalah

Berdasarkan latar belakang yang dipaparkan, didapatkan identifikasi masalah sebagai berikut :

1. Apakah dapat dibuat pola PLBWD yang adaptif pada beragam *platform* (Java EE, NET, dan PHP)?
2. Apakah dapat diimplementasikan pola PLBWD yang dikembangkan tersebut untuk beragam *platoform* dan teknologi?

1.3 Tujuan Penelitian

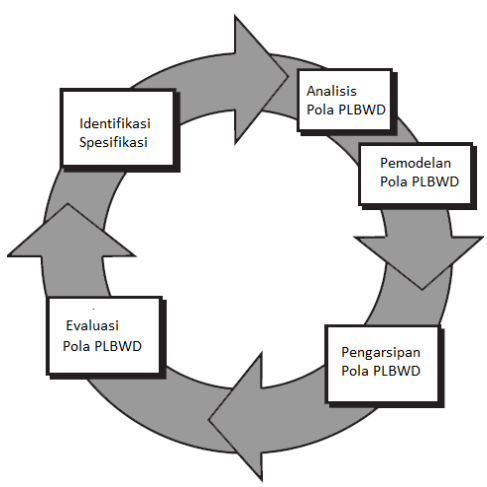
Berikut ini merupakan tujuan dari penelitian yang dilakukan:

1. Memodelkan pola PLBWD yang adaptif terhadap berbagai *platform*.
2. Membuat generator yang mampu membangkitkan kode program dari pola PLBWD yang dikembangkan?

II. ANALISIS DAN PERANCANGAN POLA PLBWD

2.1. Analisis Penyelesaian Masalah

Untuk menyelesaikan analisis pola PLBWD, dibutuhkan acauan untuk mendefiniskan tiap fase yang pada akhirnya sampai pada hasil analisis. Pada penelitian ini, acuan yang akan digunakan adalah mengadaptasi model proses *Prototyping*. Model proses *protyping* merupakan model proses yang berdiri sendiri, tetapi dapat digunakan untuk konteks tertentu dimana masih belum ada pemahaman yang jelas dari apa yang hendak dikembangkan (Pressman [4]). Berikut representasi model proses *Prototyping* pada Gambar 1.



Gambar 1
Model proses hasil adaptasi dari *Prototyping* [4].

Dari representasi pada Gambar 1 di atas, berikut paparan dari tiap fase :

1. Identifikasi Spesifikasi
   Mengenali dan berinteraksi dengan aplikasi untuk mendapatkan spesikasi, arsitektur, teknologi, dan perilaku. Mengidentifikasi modul, paket, kelas, dan *library* yang pada iterasi selanjutnya akan dianalisis lebih dalam.
2. Analisis dan Pemodelan Pola PLBWD
   Menganalisis kode program dan elemen dari aplikasi. Kode program dianalisis, direkam dan dimodelkan meski belum dipahami secara utuh. Pola utuh didapatkan pada iterasi-iterasi selanjutnya.
3. Pengarsiapan Pola PLBWD
   Melakukan inventarisir pada kode-kode aplikasi yang dianalisis. Kode tesebut dikomparasi dengan kode pada aplikasi lain. Komprasi dari kode-kode tersebut diabstraksi untuk mendapatkan pola PLBWD yang bersifat generik.
4. Evaluasi Pola PLBWD
   Mengecek hasil analisis pada iterasi sebelumnya, kemudian mengindetifikasi dan menggabungkan dengan hasil analisis terbaru. Mencocokan pola generik terbaru dengan literatur.

2.2. Resume Hasil Analisis Pola PLBWD untuk Tiap Lapisan MVC

Berikut resume hasil analisis dari tiap lapisan MVC, direpresentasikan pada Tabel 1, 2 dan 3.

Tabel 1
Lapisan View

No	View	JSF	Razor	YII
1	*Script language*	JSF	Razor	PHP

No	View	JSF	Razor	YII
2	HTML support	Ya	Ya	Ya
3	Direktori View layer	*Web*Content	Views	views
4	*Tempale Framework*	Facelete	Razor	Twig
5	Validasi	Ya	Ya	Ya
6	Ekstensi *default*	.html	.cshtml	.php
7	Penerapan *Clean code*	Ya	Tidak	Tidak
8	Direktori penyimapanan aset/source (CSS, JavaScript)	*Web*Content/ CSS *Web*Content/ Script	Content Scripts	*web*/css *web*/assets
9	*Template* anatarmuka	WEB-INF/Template	Views/Shared	Twig
10	Teknologi interaksi antarmuka dengan controller	Expresi Language (EL)	@ *character*	-
11	Internationalizati dan localization	Ya	Ya	Ya

Pada Tabel 1 dipaparkan hasil analisis dari lapisan *view* untuk semua *platform*. Dapat dilihat detail implementasi, pendekatan, dan teknologi yang digunakan berbeda-beda. Selanjutnya dari perbedaan tersebut akan abstraksi untuk mendapatkan pola yang adaptif untuk lapisan *view* dan interaksi dengan *controller*.

Tabel 2

Lapisan Controller

No	View	JSF	Razor	YII
1	*Framework* Alternatif	Spring MVC, Struts2, dan lain-lain	-	Laravel, CI, Cake PHP, dan lain-lain
2	Bahasa	Java	C#	PHP
3	Paradigma	*Obejact Oriented*	*Object Oriented*	*Object Oriented*
4	Direktori lapisan *controller*	WEB-INF/lib	Controllers	controllers
5	*Front controller*	FacesServlet.java	RouteConfig.cs	YII*ApplicationController*.php
6	Validasi antarmuka (*return*)	Ya	Ya	Ya
7	Penaganan konfigurasi aplikasi	*web*.xml	AuthConfig.cs, BundleConfig.cs, FilterConfig.cs RouterConfig.cs *web*ApiConfig.cs	Config.php
8	Direktori *libary*	WEB-INF/lib	packages	vendor
9	Konfigurasi navigasi halaman	WEB-INF/faces-config.xml	Views/*Web*.config.xml	Pagination.php
10	Variable global	Tidak	Ya	Ya

Pada Tabel 2 dipaparkan hasil analisis dari lapisan *controller* untuk semua *platform*. Lapisan *controller* yang dipilih pada peneltian ini untuk tiap *platform*, yaitu Java EE menggunakan JSF, .NET menggunakan Framework .NET, dan PHP menggunakan YII. Sama dengan lapisan *view*, pada lapisan *controller* detail implementasi, pendekatan, dan teknologi yang digunakan berbeda-beda, karena dibangun dengan *platform* dan pendekatan yang berbeda. Selain

mengabstraksi pada lingkungan *controller,* diperhatikan juga keseragaman dari mekasnisme interaksi dengan lapisan lain. Keseragaman interaksi antara *controller* dengan lapisan lain menggunakan pendekatan *interfacing*. *Interfacing* mendefiniskan interaksi secara seragam dan tidak terbebas dari spesikasi *platform* dan teknologi (Hao [5]).

Tabel 3

Lapisan Model

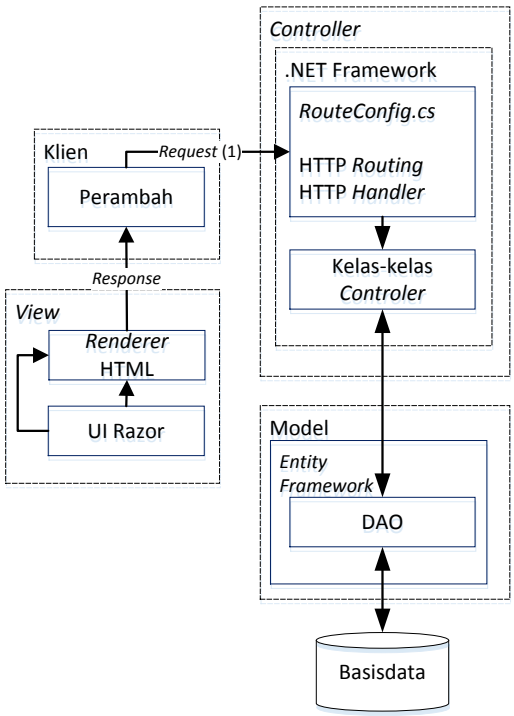
No	View	JSF	Razor	YII
1	Alternarif *Framework*	Eclipselink, Ibatis, dan lain-lain	DAO *Framework*	Eloquent
2	Bahasa	Java	C#	PHP
3	Paradigma	*Obejact Oriented*	*Object Oriented*	*Object Oriented*
4	Direktori lapisan model	Java Resources/ src	Models	models
5	Penerapan *Persistence*	Native	Native	*Third party*
6	Konfigurasi aplikasi dengan *database*	Src/hibernate.cfg.xml	*Web*.config .xml	config/db .php
7	Deklarasi Anotasi	@entity	[HttpPost]	

Pada Tabel 3 dipaparkan hasil analisis dari lapisan model untuk semua *platform*. Detail implementasi, pendekatan, dan teknoogi yang digunakan berbeda-beda. Pada *platform* Java EE dan .NET lapisan model menggunakan *framework* spesifik, yaitu Hibernate dan Entity Framework, Sedangkan pada PHP tidak menggunakan *framework*, lapisan model dikelola secara terintegrasi oleh teknologi YII. Jika menggunakan *framework* mandiri, maka memiliki efek, yaitu adanya mekanisme konfigurasi untuk menjembatani spesifikasi *controller* dengan model. Pola PLBWD yang dikembangkan harus mampu adaptif terhadap berbagai *framework* dari model yang mandiri tersebut. Keunggulan dari menggunakan *framework* adalah efisiensi dan produktivitas pengembangan dan pemanfaatan kemampuan dari *framework* tersebut untuk berbagai hal terkait penanganan data dan interaksi antara aplikasi dengan basis data (Xia [6]).

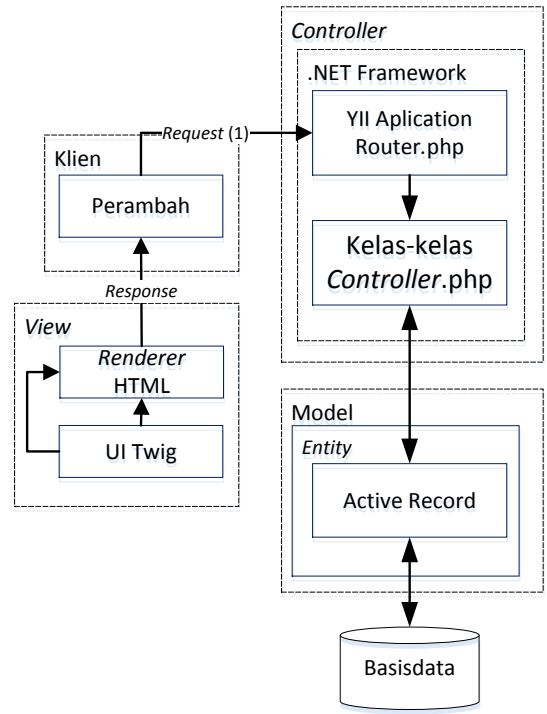
2.3 Rancangan Pola PLBWD

Fase berikutnya adalah fase perancangan dari pola PLBWD berbasis MVC untuk tiap *platform*. Fase ini dilakukan dengan membuat model arsitektur PLBWD dari tiap *platform*. Deskripsi perancangan dilakukan dengan memodelkan arsitektur berdasarkan hasil analisis. Dikembangkannya model dari tiap *platform* PLBWD dapat membantu produktivitas pemrogram ketika harus pindah dari satu *platform* ke *plarform* lain. Berdasarkan hasil analisis, dari sudut pandang abstrak, model arsitektur PLBWD yang menerapkan MVC memiliki arsitektur dan alur proses yang identik. Berikut rancangan pola PLBWD untuk tiap *platform*.

1. Model konseptual Pola PLBWD pada .NET



Gambar 2
Model konseptual Pola PLBWD pada .NET



Gambar 3
Model Konseptual Pola PLBWD pada PHP

Dari representasi Gambar 2 dapat paparkan sebagai berikut:

a. UI Diletakan pada direktori *Views.* Secara *default* di dalam direktori *Views* terdapat direktori-direktori lain yang sama penamaannya dengan nama kelas *action* di kelas *controller*, kecuali direktori Shared. Direktori *Shared* berisi berkas-berkas UI yang menjadi *template* yang digunakan untuk semua halaman *web*.

b. Untuk berinterasksi antara *view* (*tag* Razor) dengan *controler* (Bahasa C#) menggunakan teknologi @.

c. Front *controller* berada pada lapisan *controller*. Penanganan *front controller* dilakukan oleh kelas RouteConfig.cs. RouteConfig.cs didaftrakan pada direktori App_Start. RouteConfig yang bertanggunag jawab memetakan *request* pada kelas *controller atau action dari* kelas *controller* yan sesuai. App_Start merupakan direktori yang bersisi semua konfigurasi yang dijalankan pada saat inisiasi aplikasi. Contoh pemetaan *request* dengan kelas *controller :* akademik/nilai/123040. Akademik merupakan kelas *controller,* nilai

merupakan *action, dan* 23040 merupakan id.

d. Bersifat *action centris :* dimana setiap *request* akan ditangani oleh sebuah objek *logical* berupa *Controller*.

2. Model Konseptual Pola PLBWD pada PHP

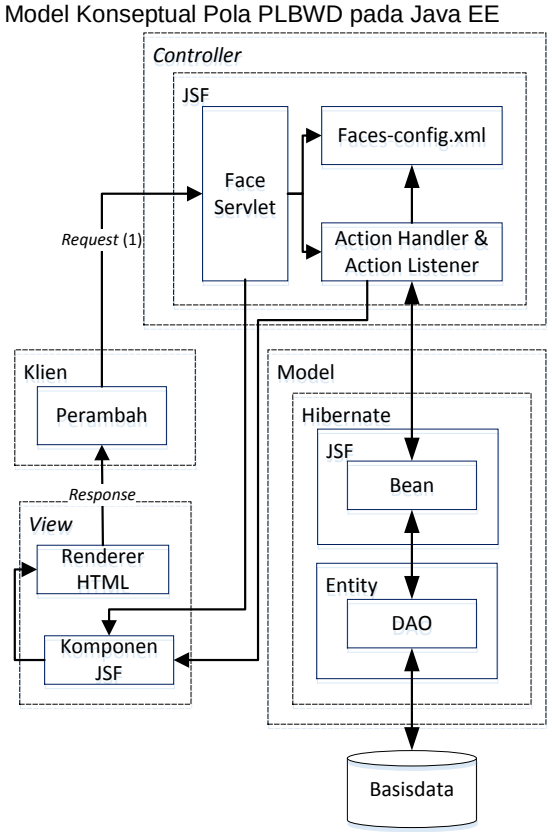
Dari representasi gambar 3 dapat paparkan sebagai berikut:

a. Lapisan *view* ditempatkan pada direktori *views.* YII memiliki sekumpulan *library* antarmuka UI yang siap pakai yang disebut *widget.* Untuk melakukan komunikasi dengan *controller* tidak memerlukan teknologi khusus, karena lapisan dari keduanya menggunakan bahasa PHP. Kecuali jika UI menggunakan bahasa selain PHP, seperti Java Script, maka memerlukan teknologi penghubung seperti JSON.

b. *Front controller* berada pada lapisan *controller. Front controller* ditanagi oleh Config.php. *Front controller* bertugas memetakan *request* ke kelas *controller* yang sesauai. *Front controller* pada YII bersifat *action page***.** *URL Manager berfungsi* menerima *requst* dari *Application,* secara teknis *URL Manager* yang melakukan pemetaan pada kelas *controller.*

c. Lapisan model berada pada direktori Models. Secara umum pola dasar dari kelas *entity* didefinisikan sebagai *Plain Old CLR Object* (POCO). *Visibility* atribut didefinisakan public. Akses ke basis data dapat menggunakan beragam teknik dan teknologi seperti *framework* ADO, Entiry Framework, NO SQL. Anotasi menggunakan simbol [ ]. Regitrasi basis data, didaftarkan dan dipetakan pada berkas WebConfig.cs.

3. Model Konseptual Pola PLBWD pada Java EE



Gambar 4

Model Konseptual Pola PLBWD pada Java EE

Dari representasi Gambar 4 dapat paparkan sebagai berikut:

a. *Front controller* diletakan pada direktori Web Content. UI menggunakan konsep

berbasis komponen. Setiap komponen pada JSF dapat ditangani oleh kelas *bean* secara mandiri.

b. Tidak mengijinkan penangangan logika di halaman JSF, sehingga kode dijamin kebersihannya.
c. Menggunakan template *facet* untuk penanganan *templating*.
d. Untuk berinterasksi antara *view* (JSF) dengan *bean* (bahasa Java) menggunakan teknologi Expression Language (EL).
e. *Front controller* Ditangani oleh kelas FacesServlet. FacesServlet didaftarkan pada kelas web.xml. Untuk memetakan *path requst* ditangani oleh faces-config.xml atau anotasi yang sesuai dengan kelas *bean.* Untuk dapat dikenali kelas *bean* harus dipetakan dengan halaman JSF. FacesServlet selain dapat menunjuk ke faces-config.xml, juga dapat menunjuk *ActionEvent*/*Listener* jika *requset* membutuhkan *action*.
f. Lapisan model berada pada direktori src. Identitas letak kelas *entiry* berdasarkan penamaan paket. Secara umum pola dasar dari kelas *entity* didefinisikan sebagai *Plain Old Java Object* (POJO). Interaksi dengan basisdata menggunakan teknik *Object Relation Model* (ORM). Anotasi menggunakan karakter @. Registrasi dan pemetaan basisdata ditangani oleh Hibernate.cfg.xml.

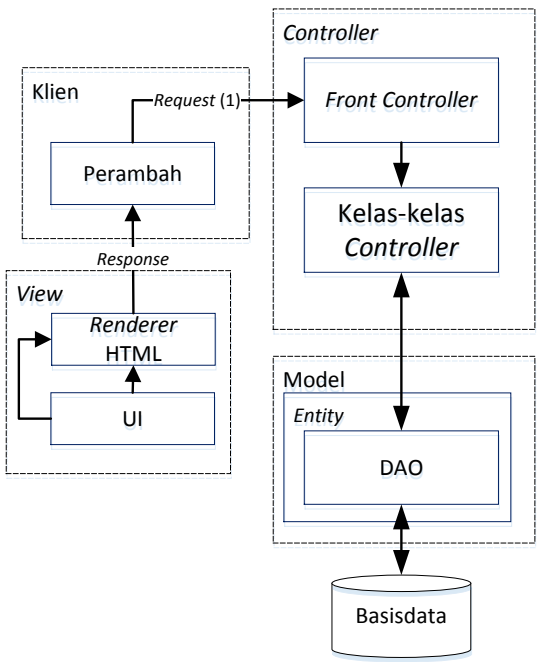
4. Model konseptual pola PLBWD

Dari paparan tiap *platform* kemudian diabstraksi untuk mendefinisikan pola pengembangan PBWD sebagai berikut :

a. Front Controller : Berada Pada lapisan *controller.* Bertanggung jawab untuk menerima *request* dari klien dan melakukan pemetaan terhadap kelas *controller*, menjalankan operasi yang diperlukan untuk menghasilkan *response.* Semua *request* dari klien harus melewati *front controller*.

b. *Controller :* Bagian yang menjembatani antara lapisan yang lain. Pola *controller* harus mampu adaptif pada keberagaman *platform* dan teknologi di lapisan yang lain.
c. UI: *View* merupakan bagian yang bertanggung jawab menangani UI. UI yang dimaksud adalah HTML yang akan ditayangkan pada sebuah perambah. *View* menangani bagaimana sebuah informasi disajikan kepada pengguna melalui perambah.
d. Model*:* Lapisan yang berinteraksi dengan basisdata. Biasanya sebuah *entity* merupakan tabel dalam basisdata relasional, dan setiap atribut *entity* sesuai dengan kolom(*field*) pada tabel. Tugas model (*entity* dan DAO) bervariasi : melakukan *query* ke basisdata, mengolah data menurut logika bisnis, dan sebagainya.

Berikut model konseptual pola umum PLBWD direpresentasikan pada Gambar 5.



Gambar 5

Model konseptual pola PLBWD

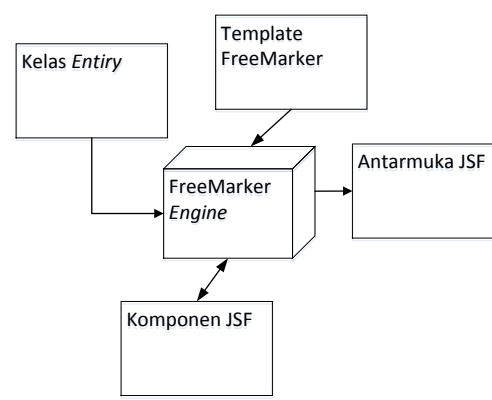
III. IMPLEMENTASI KODE GENERATOR PLBWD

3.1 Analisis dan Desain

Bagian ini, memaparkan proses analisis, perancangan hingga implementasi untuk menguji Pola MVC PLBWD dengan mengembangkan sebuah aplikasi pembangkit kode program. Aplikasi yang dikembangkan hanya dengan *platform* Java EE. Aplikasi diberi nama aplikasi Generator Antarmuka Pengguna (GAP). Aplikasi akan mampu membagkitkan lapisan UI dan *controller*, sedangkan untuk pembangkitan lapisan model akan menggunakan kakas dari *framework* Hibernate. Proses pembangkitan kelas model menggunakan *framework* Hibernate. Pembangkitan *controller* dan UI akan kembangkan sendiri.

3.2 Pembangkitan kode UI

Proses membangkitkan kode UI pada penelitian ini menggunakan *template* FreeMarker. Secara umum proses membangkitkan antarmuka menggunakan FreeMarker melibatkan *template*, kelas *entity*, dan konfigurasi FreeMarker. Proses pembangkitan UI menggunakan FreeMarker direpresentasikan pada Gambar 6.



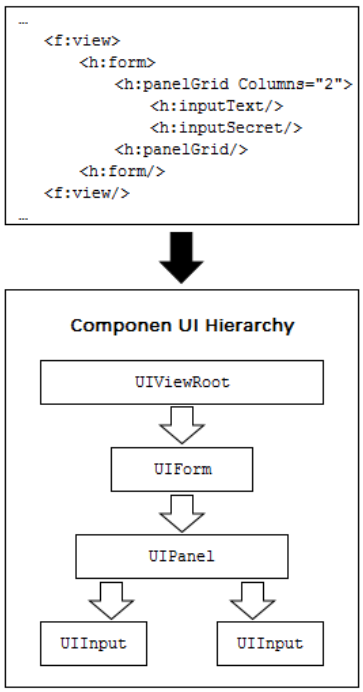
Gambar 6

Arsitektur Pembangkit kode program

3.3 Kompoen UI JSF

Komponen JSF bersifat *extensibe,* ini memungkinkan komponen JSF untuk dapat dikembangkan. Pengembang dapat menambahkan fitur-fitur yang dibutuhkan. Contoh penerapan komponen antarmuka pengguna JSF pada sebuah halaman *facelets*

beserta hirarki dari komponen UI yang diciptakan, direpresentasikan pada Gambar 7.



Gambar 7
Contoh alur pemrosesan kode UI pada JSF

Terlihat bahwa komponen <inputText> dan <inputSecret> berasal dari kelas komponen yang sama yaitu Kelas UI Input. Namun, yang membedakan adalah kelas yang me-*rendering* komponen tersebut, dimana komponen <InputText> di-*rendering* oleh kelas htmlInputText, sedangkan komponen <inputSecret> direnderer oleh kelas HtmlInputSecret.

Pada peneltian ini terdapat dua fungsi utama dari kelas *entity*, yaitu kelas *entity* sebagai 1) representasi basisdata: Kelas *entity* merupakan kelas yang mewakili tabel pada basisdata relasional, setiap penciptaan objek dari kelas *entity* merupakan satu baris data pada sebuah tabel, dan setiap atribut pada kelas *entity* merupakan atribut pada sebuah tabel. Untuk membuat sebuah kelas *entity*, pada penelitian ini menggunakan fungsi yang dimiliki oleh *framework* Hibenate, yaitu Hibernate *Mapping Files* and POJOs *from Database*, fungsi ini menghasilkan kelas *entity* yang didapat dari hasil pemetaan dengan tabel yang ada pada basisdata*; 2)* pengikat Komponen JSF : Mengikat komponen JSF dengan kelas *entity* menciptakan keterhubungan antara halaman persentasi dengan lapisan model. Mengikat komponen UI dengan kelas *entity* dapat diartikan bahwa setiap pengubahan nilai yang terjadi pada komponen UI akan berpengaruh atau dipengaruhi oleh kelas *entity* yang terkait. Untuk dapat megikat komponen JSF dengan kelas *entity* adalah dengan melalui kelas *managed bean*, dimana kelas *managed bean* tersebut memiliki atribut bertipe kelas *entity*. *Managed bean* adalah kelas Java yang terdaftar pada JSF, hanya kelas yang terdaftar sebagai *managed bean* yang dapat berinterkasi secara langsung dengan halaman JSF. *Managed bean* dapat berisi *method getter* dan *setter*, logika bisnis atau bahkan *backing bean*.

IV. PENGUJIAN APLIKASI

Setelah apliksai GAP dibuat, maka akan dilakukan pengujian pembangkitan kode UI menggunakan studi kasus. Pengujian aplikasi dilakukan dengan uji kesesuaian (*comformance*) antara rancangan pola PLBWD dengan hasil pembangkitan kode program dari aplikasi GAP. Pengujian dilakukan dengan membangkitkan kode program dari basisdata. Setelah dibangkitkan kode program oleh GAP, selanjutnya dilihat arsitektur aplikasi hasil pembagkitan tersebut dengan arsitektur perancangan yang telah dibuat.

Tampilan UI hasil pembangkitan GAP, direpresentasikan pada Gambar 8.

Name	Addressline1	City
Jumbo Eagle Corp	111 E. Las Olivas Blvd	Fort Lauderdale
New Enterprises	9754 Main Street	Miami
Wren Computers	8989 Red Albatross Drive	Houston
Small Bill Company	8585 South Upper Murray Drive	Alanta
Bob Hosting Corp.	65653 Lake Road	San Mateo
Early CentralComp	829 E Flex Drive	San Jose
John Valley Computers	4381 Kelly Valley Ave	Santa Clara
Big Network Systems	456 444th Street	Redwood City
West Valley Inc.	88 Northsouth Drive	Dearborn
Zed Motor Co	2267 NE Michigan Ave	Dearborn
Big Car Parts	52963 Notouter Dr	Detroit

Gambar 8
UI hasil pembangkitan GAP.

V. EVALUASI

Berdasar studi kasus yang telah dipaparkan sebelumnya, menunjukkan bahwa untuk dapat mengetahui pola PLBWD dengan kakas GAP, maka harus dilakukan pembangkitan kode program terlebih dahulu dari basisdata. Setelah kode program dibangkitkan, selanjutnya dilakukan komparasi pola PLBWD hasil perancangan dengan pola PLBWD hasil pembangkitan GAP. Cara membandingkan arsitektur dari keduanya adalah dengan cara melihat struktur *project* masing-masing dan melihat secara rinci kode program hasil pembangkitan GAP.

VI. KONKLUSI DAN PENELITIAN LANJUTAN

Berdasar pembahasan pada bab-bab sebelumnya dapat ditarik beberapa simpulan sementara dalam penelitian ini, yaitu :

1. Telah dihasilkan model pola PLBWD untuk tiga *platform* (Java EE, .NET, PHP) dari hasil analisis, sehingga pola yang dihasilkan dapat membantu pemrogram pada saat harus berpindah *platform*.
2. Telah dibuat aplikasi GAP pada *platform* JAVA EE menggunakan teknologi JSF untuk menguji pola PLBWD yang telah didefiniskan. PLBWD yang dihasilkan dari aplikasi GAP sudah dievaluasi dengan cara mencocokan pola PLBWD pada hasil perancangan dengan pola PLBWD yang dihasilkan dari aplikasi GAP.

DAFTAR PUSTAKA

[1] Burns, E. and Schalk. C., JavaServerFaces 2.0, New York: McGraw-Hill, 2009.

[2] Caytiles. R. D., and Lee. S., "A Review of an MVC Framework based Software Development," *International Journal of Software Engineering and Its Applications,* vol. 8, pp. 213 - 220, 2014.

[3] Jiang. Y., Cheng, Albert, Zou and Xingliang, "Schedulability Analysis for Real-Time P-FRP Tasks under Fixed Priority Scheduling," *Embedded and Real-Time Computing Systems and Applications (RTCSA) IEEE,* 2015.

[4] Pressman. R. S., Software Engineering: A Practitioner Approach 7th Edition, New York: McGraw-Hill., 2012.

[5] Hao. H. M., and Jaafar. A., "Tracing User Interface Design Pre-requirement to Generate Interface Design Specification," *Electrical Engineering and Informatics, 2009,* vol. 1, pp. 287 - 292, 2009.

[6] Xia. C., Yu. G., and Tang. M., "Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate," *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on ,* pp. 1-3, 2009.