IMPLEMENTASI *FRAMEWORK* MODEL-*VIEW*-CONTROLLER PADA SISTEM INFORMASI AKADEMIK UNIVERSITAS LAMPUNG

Kurnia Muludi[1)]

[1)]Jurusan Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam
Jl. Prof. Dr. Soemantri Brodjonegoro No. 1 Bandar Lampung 35145
Surel: kmuludi@yahoo.com

ABSTRACT

Information System characteristics should be able to handle mass transaction, could present real time data and provide interactive experince to its *user*. This paper would discuss Model-*View*-Controller framework implementation on Academic Information System of The University of Lampung. Result showed Model-*View*-Controller architecture could seperate data core processing and it functionality, consistency among *view*s, fleksible interface in relation to the *view*s and could present the information in many forms.

Keywords: Academic Information System, Model-*View*-Controller, Web Framework.

ABSTRAK

Kebutuhan Sistem Informasi yang kompleks yang dapat melayani transaksi masal, mempresentasikan data secara *real-time* dan menyajikan pengalaman interaktif ke *user*merupakan tuntuan masa kini. Dalam paper ini dibahas implementasi *Model-View-Controller framework* dalam pengembangan aplikasi web Sistem Informasi Akademik di Universitas Lampung. Hasil menunjukkan arsitektur *Model-View-Controller* yang diterapkan pada Sistem Informasi Akademik dapat memastikan pemisahan pemrosesan data inti dan fungsionalitasnya, menjaga konsistensi di antara *view-view*, pendekatan antarmuka yang fleksibel, terdapat fleksibelitas dalam menangani berbagai *view*, dan dapat merepresentasikan informasi dalam berbagai bentuk.

Kata kunci: *Model-View-Controller*, Siakad, *Web Framework*

PENDAHULUAN

Saat ini situs web semakin kompleks dalam melayani transaksi, mempresentasikan data secara real-time dan menyajikan pengalaman interaktif ke *user*. Perangkat lunak untuk web juga menjadi lebih powerful dan menyediakan fungsionalitas yang tinggi dan kompleks yang dalam pengembangannya juga dapat melibatkan pengembang secara masal dan perangkat pendukung yang beragam. *Web*

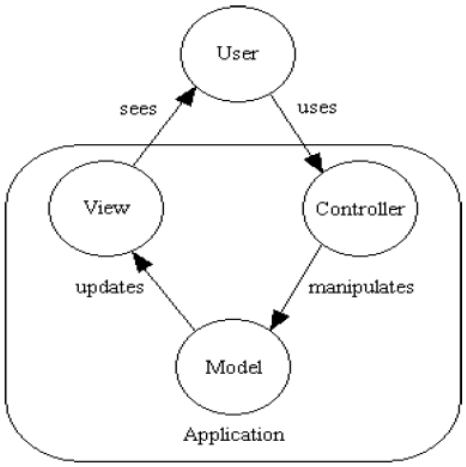
framework dalam hal ini dapat menjadi solusi yang menjanjikan karena menyediakan cara pengembangan aplikasi mulai dari nol hingga *content management system* yang tinggal pakai (*out-of-the-box* CMS). Penggunaan CMS seperti ini dapat diibaratkan membeli rumah yang sudah jadi. Kita memang dapat mengecat dan mengatur furnitur sesuai keiinginan, namun kita tidak punya pilihan selain menerima bentuk/arsitektur rumah seperti adanya. Sedangkan membangun dari nol memberi pilihan luas mulai dari memilih bentuk dan bahan yang digunakan. Dengan menggunakan web framework dalam hal pengembangan aplikasi web memungkinkan kita memiliki pilihan seperti memilih tipe dinding, kabel, sistem drainase, jendela dan lainnya yang dapat disusun sesuai dengan kebutuhan (Scot, 2008).

Perkembangan open source yang sangat marak diantaranya disebabkan keterlibatanmasal para pengembang dari segala penjuru dunia. Kode yang ditulis dapat dilihat oleh komunitas dan karenanya kode didokumentasikan dengan baik. Tool-tool open source juga mencegah keterikatan terhadap suatu vendor tertentu saja. Tool open source biasanya memiliki komunitas yang besar yang secara cepat dapat berekasi terhadap masalah yang timbul dan juga menyediakan dukungan solusi secara bersama. Dalam paper ini dibahas implementasi MVC framework dalam pengembangan aplikasi web Sistem Informasi Akademik di Universitas Lampung.

Sistem Informasi Akademik Universitas Lampung (Siakad) merupakan sistem informasi *online* yang melayani transaksi akademis antara dosen, mahasiswa, dan administrasi pendidikan di Universitas Lampung. Siakad pertama dikembangkan pada tahun 1999 hasil perpindahan sistem *offline* berbasis kertas scan yang sebelumnya telah berjalan dua dekade. Pada tahap awal ini sistem menggunakan spropreitary *software database Oracle* dan *Oracle Form*. Tahun 2000 dikembangkan lebih lanjut mengunakan

server aplikasi *ColdFusion* dan database tetap *Oracle*. Dengan meningkatnya popularitas *Server* aplikasi PHP, tahun 2005 dilakukan penulisan ulang skrip aplikasi dalam PHP namun database tidak berubah masih menggunakan *Oracle*. Meningkatnya jumlah pengguna terutama mahasiswa, menyebabkan peningkatan kebutuhan *resource/hardware* seperti *server* juga meningkat. Keterbatasan *Licence database* yang ada tidak memungkinkan database untuk secara maksimal memanfaatkan kapasitas server database yang baru. Tahun 2013 diambil keputusan menulis ulang *script* aplikasi dalam *framework open source Model-View-Controller* dan migrasi *database* ke *open source PostGre Sql*.

Secara umum pola arsitektur menyediakan notasi struktural bagi sistem perangkat lunak dan menetapkan peran tiap komponen, serta mengarahkan fungsionalitas dalam sistem (Buschmann et al., 2001). Namun pada pola arsitektural *Model-View-Controller* sistem *software* dibagi menjadi tiga sub-sistem yaitu *model*, *view* dan *controller*. Arsitektur *Model-View-Controller* (MVC) pada dasarnya diterapkan pada sistem perangkat lunak interaktif (Wikipedia, 2015).



Gambar 1. Pola arsitektur Model-*View*-Controller (Wikipedia, 2015).

Komponen *Model* berisi fungsionalitas inti dan data, komponen *View* memberikan informasi ke *user* dan Komponen *Controller* menangani masukan *user* menggunakan sistem validasi tertentu. Komponen *View* dan *Controller* bersama-sama membentuk *user interface*. Mekanisme propagasi-perubahan memastikan konsistensi antara antarmuka dan model.

Dalam tulisan ini dibahas hasil implementasi arsitektur *Model-View-Controller* (MVC) pada pengembangan Sistem Informasi Akademik Universitas Lampung.

METODE

Desain Sistem

Diagram Class-Responsibility-Collaborator Cards (Beck dan Cunningham, 2012) untuk Model-*View*-Component disajikan pada Tabel 1, 2, dan 3.

Model:

Komponen Model pada MVC men-enkapsulasi data dan funsionalitas dan independen terhadap perilaku input dan output dari sistem interaktif.

Tabel 1. Class-Responsibility-Collaborator Card untuk Komponen Model

Class: • *Model*	***Collaborators:*** • *View* • *Controller*
Tanggung Jawab: • Menyediakan fungsionalitas inti dan data • Menginformasikan komponen-komponen tentang modifikasi data	

View:

Komponen *view* bertugas menampilkan informasi ke *user*, baik dalam bentuk *view* tunggal atau jamak yang berasosiasi dengan arsitektur MVS tunggal. Komponen ini bersifat dinamis dan berasosiasi dengan komponen Controller.

Tabel 2. *Class-Responsibility-Collaborator Card* untuk Komponen *View*

Class: • *View*	**Collaborators:** • Model • Controller
Tanggung Jawab: • Menampilkan informasi ke *user* • Mengambil data dari model • Implemtasi prosedur update data	

Controller:

Komponen *Controller* selalu mengikuti mekanisme propagasi perubahan, jika *user* merubah *Controller* pada satu *view*, seluruh *view* yang berasosiasi dengannya harus merefleksikan perubahan tersebut. Sifat ini merupakan sifat dinamis MVC.

Tabel 3. *Class-Responsibility-Collaborator Card* untuk Komponen *Controller*

Class: • *Controller*	***Collaborators:*** • *Model* • *View*
Tanggung Jawab: • Menerima input/respon *user* • Menterjemahkan event *user* menjadi request ke model • Menjadi perilaku dinamis pada syncronized *view*	

Implementasi

Implementasi Arsitektur MVC dapat diterapkan untuk Sistem Informasi Akademik Uuniversitas Lampung (Siakad). Pendekatan yang dilakukan dijelaskan di bawah ini:

Konteks

Desain aplikasi interaktif dengan *multiple view* dan tersinkronisasi. Perangkat lunak harus memiliki antarmuka dan yang fleksibel terhadap berbagai kelompok pengguna.

Masalah

Masalah yang sering muncul dalam sistem informai akademik diantaranya adalah (1) mesti dapat mendukung jumlah *concurent user* yang besar pada masa-masa tertentu,(2) Informasi harus dapat ditampilkan dalam format yang berbeda-beda, dan (3) perubahan yang diterapkan pada antarmuka harus bersifat fleksibel dan mudah.

Solusi

Solusi dari permasalahan di atas adalah dengan membagi aplikasi menjadi tiga bagian yaitu input, proses dan output. Tiap bagian bersifat independen satu sama lainnya tetapi harus dapat disinkronisasi. Arsitektur *Model-View-Controller* merupakan solusi permasalahan di atas. Model berisi data dan fungsionalitas inti. Bagian Model bersifat independen terhadap representasi keluaran dan input data. *View* bertanggung jawab terhadap keluaran yang diberikan ke *user*. Dalam hal ini dimungkinkan suatu informasi memiliki *multiple view*. Bagian Controller menangani inputan *user*. Tiap *view* memiliki komponen individu yang berasosiasi dengannya.

Contoh: Pembagian Model-*View*-Controller dalam Siakad disajikan dalam Tabel 4.

Tabel 4. Pembagian MVC dalam Siakad.

MODEL	VIEW	CONTROLLER
Entitas: Mahasiswa KRS KHS Jadwal Transkrip Kurikulum Mata_kuliah Action: Login Logout Edit Add Update	Login Login_Failed Logout Edit Add Display Edit_sukses Add_sukses Delete_sukses Edit_gagal Add_gagal Delete_gagal	Login_action • If M:Sukses, M:Display • If M:gagal, M:Login_failed Edit_action • If V:Submit, M:Edit • If M:Edit_gagal, V:Edit • If M:Edit_sukses, V:Edit_sukses Add_action • If V:Submit, M:Add • If M:Add_gagal, V:Edit • If M:Add_sukses, V:Add_sukses

MODEL	VIEW	CONTROLLER
Display Rule: Mahasiswa Dosen Kajur Kaps WD1 BAAK Admin		Delete_action • If V:Submit, M:Delete • If M:Delete_gagal, V:Edit • If M:Delete_sukses, V:Delete_sukses

HASIL DAN PEMBAHASAN

Keunggulan dari implementasi arsitektur Model-*View*-Controller ada beberapa, yaitu:

1. *Multiple View* dapat diterapkan dalam Model yang sama

   Dengan dipisahnya Model dengan *View* pada arsitekture MVC, multiple *view* dapat dibuat dari sebuah model. Pada Siakad, sebagai contoh, suatu jadwal perkuliahan dapat memiliki *view* beragam tergantung kelompok *user* yang mengaksesnya. Arsitektur MVC memungkinkan kita untuk mebuat multiple *view* melalui saling terkaitnya Model dan Controller.

2. Kemudahan menjaga *View* yang tersinkronisasi:

   Mekanisme propagasi-perubahan pada Model menjamin seluruh bagian yang mengaksesnya terinformasikan mengenai perubahan yang terjadi pada saat yang sama. Hal ini mensinkronkan semua *View* dan Controler yang mengacunya. *View* yang tersinkroniasi menyederhana proses dalam Siakad.

3. Bongkar-pasang *View* dan controller.

   Pemisahan Model-*View*-Controller memungkinkan *view* dan controller dari suatu model dapat dengan mudah dibongkar pasang. Obyek antar-muka bahkan dapat disubstitusi pada saat run-time untuk berbagai stakeholder.

4. Pendekatan Antar-muka fleksibel

   Pendekatan antar-muka fleksibel merujuk pada pembuatan macam-macam *View* sesuai perspektif pengguna. Pendekatan ini meningkatkan pemahaman pengguna akan antarmuka pada berbagai model bisnis.

Catatan implementasi Model-*View*-Controller adalah sebagai berikut:

1. Kompleksitas

   Kompleksitas merupakan tingkat kesulitan dalam pemahaman model dan pembuatannya. Arsitektur Model-*View*-Controller tidak selalu cocok untuk semua model aplikasi interaktif. Desain code untuk akses ke data dan fungsionalitas controller dan model bisa saja sangat kompleks.

2. Pengendalian *update*

   Jika suatu antarmuka berhubungan kepada banyak *update*, komponen Model seharusnya dapat melompati/mengabaikan notifikasi perubahan yang tidak penting.

3. Ketatnya pasangan *View*-Controller terhadap model

   Komponen *View* dan Controller berhubungan langsung dengan Model. Ini berimplikasi pada perubahan antarmuka Model akan merusak *code* baik pada *View* dan Controller. Masalah ini akan lebih terasa jika *View* dan Controller berjumlah sangat banyak.

KESIMPULAN

Arsitektur Model-*View*-Controller yang diterapkan pada Sistem Informasi Akademik dapat memastikan pemisahan pemrosesan data inti dan fungsionalitasnya, konsistensi di antara *view-view*, dan pendekatan antarmuka yang fleksibel. Terdapat fleksibelitas dalam menangani berbagai *view*, dan dapat merepresentasikan informasi

dalam berbagai bentuk. Namun demikian tujuan dari sistem yang interaktif adalah memberikan tingkat pemanfaatan yng tinggi bagi aplikasi. Dengan menggunakan arsitektur Model-*View*-Controller, *multiple view* dapat dibuat sesuai kebutuhan aplikasi.

DAFTAR PUSTAKA

Beck K & Cunningham W. 2012. *A laboratory for teaching object oriented thinking, ACM SIGPLAN Notices (india, NY, USA: Abg) 24 (10): 1–6.*

Buschmann F, Meunier R. Rohnert H, Sornmerlad P, & Stal M. 2001. Pattern Oriented Software architecture, volume 1, February 2001.

Hacker S. 2008. Notes on a Django Migration. http://birdhouse.org/blog/2008/11/19/ notes-on-a-django-migration. [19 November2008].

Wikipedia. 2015. Model-*View*-Controller http://en.wikipedia.org/wiki/Model-*view*-controller.