JUSTINDO
Jurnal Sistem dan Teknologi Informasi
http://jurnal.unmuhjember.ac.id/index.php/JUSTINDO
E-ISSN: 25541-5735, P-ISSN: 2502-5724
Volume 6, No. 2, Agustus 2021

Penerapan *Pattern* MVC (*Model View Controller*) dalam Pengembangan Aplikasi Identifikasi Jam Puncak Arus Lalu Lintas pada Simpang Lima

Muhammad Alkaff[1], Iphan Fitrian Radam[2], Winarto Chandra[3]

Universitas Lambung Mangkurat
Email: m.alkaff@ulm.ac.id

(Naskah masuk: 4 Maret 2021, diterima: 1 Juli 2021, diterbitkan: 30 Agustus 2021)

ABSTRAK

Architecture patterns merupakan panduan dan petunjuk untuk membuat suatu aplikasi agar maintainable dan dapat dikembangkan dengan mudah, serta mempercepat proses pengembangan dengan menyediakan cara yang telah terbukti untuk menyelesaikan masalah yang terjadi dikemudian seperti yang dilakukan Model-View-Controller (MVC). Tujuan penelitian ini untuk membangun sistem yang dapat menyederhanakan proses kalkulasi dan sortir untuk menemukan jam puncak arus lalu lintas pada suatu simpang lima jalan raya berbasis website dengan menerapkan MVC dan melakukan pengujian code smell. Penerapan pattern MVC berhasil diterapkan pada pengembangan aplikasi identifikasi jam puncak arus lalu lintas pada suatu simpang lima jalan raya berbasis website, serta telah dilakukan pengujian code smell pada kode programnya dengan membandingkan pada kode program native (tanpa MVC). Berdasarkan hasil pengujian code smell di antara kode program menggunakan MVC dan yang tidak, didapatkan bahwa dalam 5 case MVC tidak ditemukan code smell, sedangkan kode program native ditemukan code smell pada 4 case dari 5 case yang diuji yaitu Duplicate Code, Long Method, Excessively long line of code (or God Line), Data clump, Excessively short identifiers, dan Middle Man.

***Kata kunci**: MVC, code smell, simpang lima, identifikasi jam puncak*

ABSTRACT

Architecture patterns are guidelines and intructions for making an application maintainable and easily expandable, and speed up the development process by providing a proven way to solve problems tha occur in the future as the Model-View-Controller(MVC)does.the purpose of this study is to build a system that can simplify the calculation and sorting process to find peak hours of traffic flow at a website-based intersection by implementing MVC and conducting code smell testing. The application of the MVC pattern has been succesfully applied to the development of an application to identify the peak hours of traffic flow at a website-based intersection. And a code smell test has been carried out on the program code by comparing it to the native program code (without MVC). Based on the results of the code smell test between the program code using MVC and those not. It was found that in 5 cases of MVC no code smell was found, while the native code smell was found in 4 cases of 5 cases tested, namely Duplicate Code, Long Method, Excessively long line of code (or God Line), Data clump, Excessively short identifiers, and Middle Man.

***Keyword**: MVC, code smell, intersection, peak hour identification*

1. PENDAHULUAN

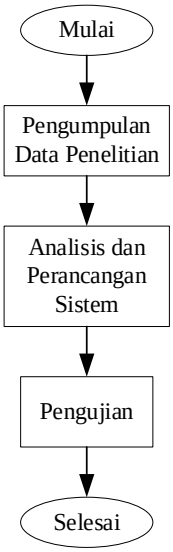
Pengembangan aplikasi biasanya terdiri dari tiga bagian kode yaitu *front-end*, *database*, dan *back-end* (Xing et al., 2019). Ketiga bagian tersebut tergabung menjadi satu bagian, namun apabila semakin banyak modul aplikasi yang akan dibuat maka *Model* ini kurang baik diterapkan, karena dapat membingungkan *programmer* dalam melakukan pengembangan aplikasi (Shylesh, 2017). Sehingga diperlukan *architecture patterns* yang bertujuan untuk membantu dalam mengidentifikasi dan spesifikasi *objects*, *class*, dan *components* (Martin, 2017; Mwendi, 2014). *Architecture patterns* merupakan panduan dan petunjuk untuk membuat suatu aplikasi agar *maintainable* dan dapat dikembangkan dengan mudah (Martin, 2017), serta mempercepat proses pengembangan dengan menyediakan cara yang telah terbukti untuk menyelesaikan masalah yang terjadi dikemudian hari (Kalawsky et al., 2013; Martin, 2017) seperti yang dilakukan *Model-View-Controller* (MVC) (Mallawaarachchi, 2020).

MVC memecah aplikasi web menjadi 3 layer, di mana bagian pertama ialah *Model* yang memiliki kaitan terhadap operasi *database*, bagian kedua yaitu *View* yang kaitannya dengan *interface*, dan bagian ketiga yaitu *Controller* yang kaitannya dengan logika aplikasi serta mengendalikan *View* dan *Controller* pada alur data (Martin, 2017). Namun, MVC lebih unggul jika dibandingkan dengan HMVC, karena HMVC sedikit lebih lambat dari MVC serta membutuhkan lebih banyak waktu (https://ellislab.com, n.d.; *Model View Controller - HMVC Pros and Cons When Compared to MVC in PHP*, n.d.).

MVC mengikuti pendekatan *Layering* di mana menjadi suatu logika yang membagi kode pad fungsi di kelas yang berbeda (Deacon, 2009; Majeed & Rauf, 2018; Martin, 2017) sehingga banyak diterapkan dalam penelitian. Andri Sunardi dan Suharjito melakukan perbandingan *Laravel Framework* dan *Slim Framework* pada MVC, di mana *Laravel* lebih baik dari sisi pengiriman serta penerimaan KB/sec (Sunardi & Suharjito, 2019). Abdul Majeed dan Ibtisam Rauf memberikan wawasan mengenai MVC untuk pengembangan aplikasi web modern (Majeed & Rauf, 2018), didapatkan MVC menjadi pilihan yang baik dalam menyediakan beberapa fitur canggih. MVC menjadikan pengembangan aplikasi web menjadi efektif dan mengurangi kompleksitas (Majeed & Rauf, 2018; Sunardi & Suharjito, 2019).

Berdasarkan penelitian-penelitian sebelumnya, MVC mampu mengurangi kompleksitas pengembangan aplikasi, sehingga dalam penelitian ini dibuat sebuah aplikasi identifikasi jam puncak arus lalu lintas pada simpang lima dengan MVC. Dengan demikian, MVC dijadikan pola dalam pengembangan aplikasi untuk mengatasi permasalahan kerumitan dalam pengolahan data jam puncak pada simpang lima jalan sebagai sumber konflik lalu lintas (Shokrolah Shirazi & Morris, 2016) (Alkaff et al., 2020) (Li, 2013) (Binoy Mascarenhas & Akhila Suri, 2018). Untuk melihat kelebihan MVC pada pengembangan aplikasi dilakukan pendeteksian *code smell*.

2. METODOLOGI PENELITIAN



Gambar 1. Alur Penelitian

Tahapan dimulai dengan identifikasi masalah dengan mencari potensi permasalahan dan bagaimana cara penyelesaian masalah tersebut. Kemudian mencari teori serta konsep yang berkaitan dengan penelitian sebagai landasan teori. Pada tahap selanjutnya yaitu pengumpulan data dan dilakukan analisis dan perancangan sistem. Tahap selanjutnya adalah melakukan implementasi sistem serta melakukan pengujian *code smell* terhadap sistem yang telah dibangun.

p-ISSN : 2502-5724; e-ISSN : 2541-5735

2.1. Pengumpulan Data

Pengumpulan data dilakukan dengan survei arus lalu lintas pada simpang lima di Kota Banjarmasin. Data tersebut untuk menganalisis dan merancangan sistem yang ingin dibuat.

2.2. Analisis dan Perancangan Sistem

Setelah mendapatkan data, maka selanjutnya melakukan analisis dan perancangan sistem. Perancangan serta pembuatan sistem dilakukan dengan menerapkan pola arsitektur MVC dengan framework Laravel dan PHP hingga sistem berhasil dibuat.

2.3. Pengujian

Pengujian dilakukan dengan pendeteksian *code smell*. Konsep *code smell* adalah struktur yang perlu dikeluarkan dari kode sumber oleh *refactoring* untuk meningkatkan kemampuan pemeliharaan perangkat lunak (Maur´ıcio et al., 2018). *Code smell* itu sendiri bukanlah masalah tapi merupakan pertanda adanya masalah. Ini menunjukkan struktur dan kualitas produk perangkat lunak yang buruk. *Code smell* tidak sama dengan kesalahan sintaks atau peringatan kompilator. *Code smell* menjadi indikasi buruknya desain program atau praktik pemrograman yang buruk yang bisa membuat proyek perangkat lunak sulit dikembangkan dan dipelihara saat program membutuhkan modifikasi. Dari beberapa parameter yang ada pada *code smell*, hanya 6 parameter yaitu *Long Parameter List*, *Large Class*, *Lazy Class*, *Feature Envy*, *Long Method*, dan *Dead Code*.

3. HASIL DAN PEMBAHASAN

3.1. Pengujian *Code Smell*

Pengujian dilakukan dengan menggunakan deteksi *code smell* pada dua program yaitu kode program php dengan menggunakan *framework Laravel* yang sudah menyediakan MVC dibandingkan dengan kode program php *native* tanpa menggunakan *framework*. Untuk hasil *code smell* yang terdeteksi dari 5 *case* yang ada dapat dilihat pada Tabel 1.

Tabel 1. Pengujian *Code smell*

Case	PHP *Native*		PHP *Laravel* (MVC)		Keterangan
	Code smell	No *Code smell*	*Code smell*	No *Code smell*	
Case 1	✓	-	-	✓	*Duplicate Code*
Case 2	✓	-	-	✓	*Long Method, Excessively long line of code (or God Line), Duplicate Code atau Data clump, Excessively short identifiers*
Case 3	✓	-	-	✓	*Excessively long line of code (or God Line), Excessively short identifiers*
Case 4	-	✓	-	✓	*Tidak ada code smell*
Case 5	✓	-	-	✓	*Middle Man*

3.1.1. *Case* 1

```
public function __construct() {
    try {
    //Connect to Mongo
    $this->mongo = new Mongo('127.0.0.1:27017');

    $this->db = $this->mongo->selectDB('trafficapps2');

    $tableName = 'simpang5';
    $this->table = $this->db->$tableName;
    }

    catch(Exception $e) {
    echo "Something Went Wrong.";
    exit();
    }
}
```

Gambar 2. Terdeteksi *Duplicate Code*

Pada *Case* 1, terdapat *code smell* berupa *duplicate code* pada fungsi *constructor* di *Native*, yang mana akan terdapat perulangan kode program di kelas lain untuk penulisan kode koneksi ke *database* menggunakan MongoDB.

3.1.2 *Case* 2

```
public function store() {
    if(isset($_POST['nama_simpang_5']) && isset($_POST['nama_jalan_utara'])
    && isset($_POST['nama_jalan_timur']) && isset($_POST['nama_jalan_selatan'])
    && isset($_POST['nama_jalan_barat']) && isset($_POST['nama_jalan_blaut'])
    && isset($_POST['ekivalen'])) {
    if($_POST['nama_simpang_5'] !== '' && $_POST['nama_jalan_utara'] !== ''
    && $_POST['nama_jalan_timur'] !== '' && $_POST['nama_jalan_selatan'] !== ''
    && $_POST['nama_jalan_barat'] !== '' && $_POST['nama_jalan_blaut'] !== ''

    && $_POST['ekivalen'] !== '') {
    $namaSimpang5 = $_POST['nama_simpang_5'];
    $namaJalanUtara = $_POST['nama_jalan_utara'];
    $namaJalanTimur = $_POST['nama_jalan_timur'];
    $namaJalanSelatan = $_POST['nama_jalan_selatan'];
    $namaJalanBarat = $_POST['nama_jalan_barat'];
    $namaJalanBlaut = $_POST['nama_jalan_blaut'];
    $ekivalen = $_POST['ekivalen'];
    $mc = $_POST['mc'];
    $lv = $_POST['lv'];
    $hv = $_POST['hv'];
    $create = array(
    'nama_simpang_5' => $namaSimpang5,
    'nama_jalan_utara' => $namaJalanUtara,
    'nama_jalan_timur' => $namaJalanTimur,
    'nama_jalan_selatan' => $namaJalanSelatan,
    'nama_jalan_barat' => $namaJalanBarat,
    'nama_jalan_blaut' => $namaJalanBlaut,
    'ekivalen' => $ekivalen,
    //Excessively short identifiers
    'mc' => $mc,
    'lv' => $lv,
    'hv' => $hv,
    );
    $this->table->insert($create);
    return 'Data Simpang created successfully.';
    }
    }
    }
```

Gambar 3. Terdeteksi *Long Method*

Dari *case* 2, *Native* yang tidak menerapkan arsitektur MVC, perlu untuk melakukan cek terhadap setiap *request* yang masuk menggunakan *isset* agar tidak terjadi error pada PHP yang memberitahukan "*undefined index*" atau nama *request* tidak diketahui. Hal ini mengakibatkan munculnya *code smell* bernama ***god-line*** yaitu terdeteksinya baris kode yang terlalu panjang. Sedangkan dari MVC, *request* dikirimkan ke *Controller* tanpa menyentuh bagian *View* sehingga tidak perlu menggunakan *isset* (tidak satu halaman).

Kemudian, dari *Native* terdapat *code smell* berupa ***duplicate code*** dalam penulisan deklarasi variabel menggunakan nilai pada *request* untuk method *store* dan *update*. Sedangkan dari MVC, karena fungsinya dalam berkolaborasi, *request* yang ditulis di *View* bisa langsung diterima oleh *Controller*, sehingga dari sisi *Controller* bisa langsung menerima semua *request* tanpa menulis ulang.

Dalam deklarasi variabel ini, dari *Native* terdapat *code smell* berupa *excessively short identifiers* atau variabel yang ditulis terlalu pendek sehingga tidak menjelaskan fungsinya. Variabel yang dimaksud di sana, yaitu: Variabel Sepeda Motor (MC), Variabel Kendaraan Ringan (LV), dan Variabel Kendaraan Berat (HV).

Dengan adanya banyak baris kode di dalam *method store* dari *Native*, tercium *code smell* berupa *long method*, di mana dapat menimbulkan pertanyaan mengapa baris kode pada *method* bisa mencapai lebih dari 10 baris.

3.1.3 *Case* 3

```
$simpang5['mc'] = $_POST['mc'];
$simpang5['lv'] = $_POST['lv'];
$simpang5['hv'] = $_POST['hv'];
```

Gambar 4. Terdeteksi *excessively short identifiers*

Pada *case* 3 didapatkan bahwa *Native* terdapat *code smell* berupa ***god line* dan *excessively short identifiers.***

3.1.4 *Case* 4

Gambar 5. Perbandingan MVC dan Native

Pada *case* 4, tidak tardapat *code smell* di antara keduanya, baik dari MVC maupun dari

p-ISSN : 2502-5724; e-ISSN : 2541-5735

Native tanpa arsitektur MVC. Tetapi, bila dilihat dalam potongan *screenshoot*, pada *Native* tidak diketahui bahwa *method destroy* itu berfungsi untuk menghapus apa karena tidak adanya pengenal (berupa variabel).

3.1.5 Case 5

```
function action(){
    $simpang5Obj = new Simpang5();
    $id = $_GET['id'];
    $action = $_GET['action'];
    if($action == 'delete' && !empty($id)) {
    $simpang5Obj->destory($id);
    } else if ($action == 'edit') {
    $simpang5Obj->update($id);
    } else {
    $simpang5Obj->store();
    }
}
```

Gambar 6. Terdeteksi *Middle Man*

Pada *case* 5 untuk *native* ditemukan **middle-man**. Untuk keseluruhan fungsi, *Native* memerlukan *middle man* (fungsi *action*) untuk melakukan eksekusi dari tiap *method* yang ada. Berbeda dengan arsitektur MVC yang mana pengiriman dan penerimaan *request* mengikuti alur arsitekturnya. Dalam hal ini, **middle-man** sendiri adalah bagian dari *code smell*.

Tidak hanya itu, ada kemungkinan terdapat *code smell* berupa ***duplicate code*** untuk semua kelas karena penggunaan *middle man* dari kelas lainnya yang mana memiliki kegunaan yang sama.

3.2. Pembahasan

Berdasarkan pengujian *code smell* didapatkan bahwa perbandingan dari kedua kode program tersebut berdasarkan 5 *case* yang ditentukan didapatkan bahwa pada PHP *Native* 4 dari 5 *case* mengandung beberapa *code smell* seperti *Duplicate Code Long Method, Excessively long line of code (or God Line), Duplicate Code atau Data clump, Excessively short identifiers*, dan *Middle Man*. Sedangkan pada PHP *Laravel* (MVC) dari 5 *case* yang diuji dan dibandingkan didapatkan bahwa tidak ada terdeteksi *code smell*. Berikut ini adalah kode program dari PHP *Native* yang terdeteksi *code smell*.

Pada pengujian yang dilakukan pada *case* 1 didapatkan bahwa pada kode program PHP *Native* mengandung *code smell* yaitu *Duplicate Code* pada *constructor* di *Native*, yang mana terdapat perulangan kode program di class lain untuk penulisan atau koneksi database di *constructor* pada setiap *class*.

Sedangkan pada pengujian yang dilakukan pada *case* 2 didapatkan bahwa pada kode program PHP *Native* mengandung beberapa *code smell* yaitu *Long Method, Excessively long line of code (or God Line), Duplicate Code atau Data clump, Excessively short identifiers*.

Untuk pengujian yang dilakukan pada *case* 3 didapatkan bahwa pada kode program PHP *Native* mengandung beberapa *code smell* yaitu *Excessively long line of code (or God Line), Excessively short identifiers*.

Namun, pada *case* 4 ketika dilakukan pengujian dengan membandingkan kode program *Laravel* dengan MVC dan *native* ditemukan bahwa tidak terdapat *code smell*.

Pengujian pada *case* yang terakhir didapatkan bahwa pada kode program PHP *Native* mengandung satu *code smell* yaitu *Middle Man*.

Sistem identifikasi jam puncak arus lalu lintas pada simpang lima yang sudah dibuat terdiri dari beberapa halaman seperti dashboard sebagai tampilan awal untuk menampilkan keterangan mengenai petunjuk penggunaan sistem. Selain itu juga terdapat halaman input data lalin, di mana pengguna dapat memilih simpang lima, arah simpang dan tanggal survei yang di-input data lalu lintasnya. Kemudian pengguna memilih file data lalu lintas dalam format Excel pada setiap arah simpang yang di-input data lalu lintasnya. Pengguna mengklik tombol Import Excel untuk meng-import data lalu lintas kemudian pengguna dapat melihat data lalin yang di-input sebelum dimasukkan ke dalam database sistem. Sistem akan memproses perhitungan lalu menampilkannya pada halaman hasil. Pada halaman tersebut pengguna dapat memilih tanggal survei untuk melihat hasil identifikasi kendaraan per-jam suatu

simpang lima yang akan ditampilkan dalam bentuk table dan dapat dicetak.

4. KESIMPULAN

Penelitian ini bertujuan untuk untuk menerapkan pola *architecture pattern* MVC pada pengembangan aplikasi identifikasi jam puncak arus lalu lintas pada suatu simpang lima jalan raya berbasis website hingga membangun sistem yang dapat menyederhanakan proses kalkulasi dan sortir untuk menemukan jam puncak arus lalu lintas pada suatu simpang lima jalan raya, yang pada akhirnya didapatkan kesimpulan sebagai berikut.

1. Penerapan pattern MVC berhasil diterapkan pada pengembangan aplikasi identifikasi jam puncak arus lalu lintas pada suatu simpang lima jalan raya berbasis website, serta telah dilakukan pengujian *code smell* pada kode programnya dengan membandingkan pada kode program *native* (tanpa MVC). Berdasarkan hasil pengujian *code smell* di antara kode program menggunakan MVC dan yang tidak, didapatkan bahwa dalam 5 *case* MVC tidak ditemukan *code smell*, sedangkan kode program *native* ditemukan *code smell* pada 4 *case* dari 5 *case* yang diuji.
2. Penggunaan aplikasi berbasis website dapat menyederhanakan proses kalkulasi dan sortir untuk menemukan jam puncak arus lalu lintas pada suatu simpang lima jalan raya, karena aplikasi yang dibangun dapat mempersingkat waktu dalam pencarian jam puncak.

5. DAFTAR PUSTAKA

Alkaff, M., Radam, I. F., & Sugiantoro, S. (2020). Rancang Bangun Sistem Identifikasi Arus Lalu Lintas pada Simpang Tiga Menggunakan Database NoSQL. *Jurnal Teknik Informatika Dan Sistem Informasi*, *6*(2). https://doi.org/10.28932/jutisi.v6i2.2567

Binoy Mascarenhas & Akhila Suri. (2018). A Study on Intersection Typology and Road Safety: Case of Mumbai. *INTERGOVERNMENTAL ELEVENTH REGIONAL ENVIRONMENTALLY SUSTAINABLE TRANSPORT (EST) FORUM IN ASIA*.

Deacon, J. (2009). Model-View Controller (MVC) Architecture. *JOHN DEACON Computer Systems Development, Consulting & Training*.

https://ellislab.com, E. (n.d.). *HMVC vs MVC's speed and other advantages | General Discussion*. ExpressionEngine. Retrieved November 14, 2020, from https://expressionengine.com/forums/archive/topic/150037/hmvc-vs-mvcs-speed-and-other-advantages

Kalawsky, R. S., Joannou, D., Tian, Y., & Fayoumi, A. (2013). Using Architecture Patterns to Architect and Analyze Systems of Systems. *Procedia Computer Science*, *16*, 283–292. https://doi.org/10.1016/j.procs.2013.01.030

Li, B. (2013). A Model of Pedestrians' Intended Waiting Times for Street Crossings at Signalized Intersections. *Transportation Research Part B: Methodological*, *51*, 17–28. https://doi.org/10.1016/j.trb.2013.02.002

Majeed, A., & Rauf, I. (2018). MVC Architecture: A Detailed Insight to the Modern Web Applications Development. *Peer Review Journal of Solar & Photoenergy Systems*.

Mallawaarachchi, V. (2020, September 2). *10 Common Software Architectural Patterns in a nutshell*. Medium. https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design* (1st ed.). Prentice Hall Press.

Maur´ıcio, A., Gabriele, B., Christoph, T., Marco, A. G., & Arie, van D. (2018). Code Smells for Model-View-Controller Architectures. *Empir Software Eng*.

model view controller—HMVC pros and cons when compared to MVC in PHP. (n.d.). Stack Overflow. Retrieved November 14, 2020, from https://stackoverflow.com/questions/9342145/hmvc-pros-and-cons-when-compared-to-mvc-in-php

Mwendi, E. (2014). Software Frameworks, Architectural and Design Patterns. *Journal of Software Engineering and Applications*, *07*, 670–678. https://doi.org/10.4236/jsea.2014.78061

Shokrolah Shirazi, M., & Morris, B. (2016). Looking at Intersections: A Survey of Intersection Monitoring, Behavior and Safety Analysis of Recent Studies. *IEEE Transactions on Intelligent Transportation Systems*, *PP*, 1–21. https://doi.org/10.1109/TITS.2016.2568920

Shylesh, S. (2017). *A Study of Software Development Life Cycle Process Models* (SSRN Scholarly Paper ID 2988291). Social Science Research Network. https://doi.org/10.2139/ssrn.2988291

Sunardi, A. & Suharjito. (2019). MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based. *Procedia Computer Science*.

Xing, Y., Huang, J., & Lai, Y. (2019). *Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development*. 68–72. https://doi.org/10.1145/3313991.3314021