Implementasi *Clean Code* pada Pengembangan Aplikasi Berbasis Web

Fajri Idza Inayah
Program Studi Sarjana Informatika
Universitas Islam Indonesia
Yogyakarta, Indonesia
17523122@students.uii.ac.id

Moh. Idris, S.Kom, M.Kom
Program Studi Sarjana Informatika
Universitas Islam Indonesia
Yogyakarta, Indonesia
moh.idris@uii.ac.id

Abstract*—Kerjasama tim adalah sebuah kemampuan yang harus dimiliki oleh suatu kelompok dalam mencapai tujuan yang sama. *Programmer* merupakan sebuah profesi yang bertanggung jawab dalam mengembangkan suatu sistem aplikasi. *Programmer* menuliskan kode program (*Syntax*) dan merancang sistem untuk mengembangkan suatu aplikasi. Seorang *programmer* harus memiliki kemampuan untuk berkolaborasi dengan tim untuk membuat suatu sistem. Untuk meningkatkan kemampuan *programmer* dalam kerjasama tim, *programmer* harus memahami kode yang dituliskan oleh *programmer* yang lain. Dengan konsep *Clean Code, programmer* dapat menuliskan sebuah kode yang mudah dibaca dan dimengerti untuk berkolaborasi dengan tim. *Programmer* perlu memperhatikan kode yang ditulis, menulis kode yang buruk akan berdampak ketika terjadi perubahan dan *maintenance*. Hal tersebut dapat merugikan tim karena kode yang dituliskan buruk, sulit dibaca dan dipahami. Dengan menerapkan konsep *clean code*, kode akan menjadi lebih rapi, mudah dipahami dan terstruktur. Pada makalah ini akan dibahas tentang implementasi *clean code* pada pengembangan aplikasi berbasis web. Hasil implementasi *clean code* dapat membantu meningkatkan efektifitas serta efisiensi dalam melakukan pengembangan aplikasi serta memudahkan pengembang ketika terjadi perubahan terhadap kode.

Keywords—clean code, programmer, sistem, kode, dekomposisi

I. PENDAHULUAN

Programmer merupakan sebuah profesi yang bertanggung jawab dalam mengembangkan suatu sistem aplikasi. *Programmer* menuliskan kode program (*Syntax*) dan merancang sistem untuk mengembangkan suatu aplikasi[1].

Programmer mengikuti konsep teori seperti algoritma, struktur data dan aspek paradigma pemrograman yaitu orientasi objek atau pemrograman fungsional. Dalam menulis kode, terdapat sebuah konsep bernama *clean code*, dengan mengikuti konsep tersebut *programmer* dapat menuliskan kode yang bersih dan mudah dipahami oleh *programmer* lain. Sebuah studi di antara 227 profesional IT menyatakan bahwa keterampilan yang harus dimiliki seorang *programmer* yaitu kemampuan untuk membaca, memahami, dan memodifikasi kode program yang ditulis oleh orang lain. Kualitas kode yang rendah memberikan pengaruh langsung terhadap perawatan sebuah program[2].

Kode yang buruk sangat mempengaruhi kualitas kode sebuah program. Kode yang buruk akan sangat susah untuk dikembangkan karena kode yang dibuat tidak beraturan dan tidak terstruktur sehingga sulit untuk dibaca dan dipahami. Kode yang sulit dibaca dan dipahami akan menghabiskan waktu untuk memahami maksud dan tujuan dari kode tersebut[3].

Menerapkan konsep *Clean Code* sangat berpengaruh dalam mengembangkan aplikasi, untuk berkolaborasi dalam sebuah *project* yang akan dikerjakan tim setiap *programmer* harus mengikuti konsep *clean code*, sehingga dengan kode yang baik dan benar, antar *programmer* tidak mengalami kesulitan ketika melanjutkan kode dari *programmer* lain. Selain itu menerapkan *clean code* dapat menghemat waktu dan biaya saat harus dilakukan *maintenance* terhadap sistem[4].

Penelitian ini dimaksudkan untuk mengetahui cara implementasi clean code pada pengembangan aplikasi berbasis web PHP menggunakan *framework* Laravel. Diharapkan penelitian ini dapat memberikan manfaat apabila ingin menerapkan konsep *clean code* pada *project* pengembangan aplikasi.

II. LANDASAN TEORI

A. Kode Program

Kode program yaitu suatu rangkaian pernyataan atau deklarasi yang ditulis dalam bahasa pemrograman komputer yang dapat dibaca oleh manusia. Kode program merupakan rangkaian kode yang tersusun untuk menjalankan instruksi yang dapat dipahami oleh komputer[5].

B. Clean Code

Menurut Robert, *clean code* adalah kumpulan kode yang mudah dibaca, mudah dimengerti dan mudah di *maintenance* oleh pengembangnya sendiri maupun orang lain[6]. *Clean code* diperlukan karena beberapa alasan, yang pertama adalah komunikasi, dengan membuat kode yang rapi dan benar *programmer* dapat menyampaikan dengan tepat maksud dari kode atau fungsi yang dibuat. Alasan berikutnya yaitu untuk kolaborasi, kode yang baik dan benar akan memudahkan *programme*r dalam berkolaborasi karena kode mudah dipahami dan dibaca[7].

Menerapkan *clean code* dimulai dengan aturan penaaman, menghindari ambiguitas pada sebuah kondisi, menggunakan *coding style*, menghapus kode yang tidak digunakan dan dekomposisi program[8].

1) *Meaningful Names*

Penamaan yang baik sangat penting dalam menuliskan kode, baik untuk fungsi, kelas dan *variable*. Penamaan *variable* dalam komputer harus bermakna. Hal ini dapat membantu pemahaman penulis kode program. Bermakna bisa diartikan sebagai nama yang deskriptif yang menjelaskan arti dari nama tersebut[9].

Nama yang bermakna dapat memudahkan pengembang lain untuk memahami maksud dan kegunaan dari nama tersebut. Contohnya seperti memberikan nama pada *variable* untuk sebuah tanggal, int d akan lebih sulit dipahami ketika dipanggil di kode lain, akan lebih mudah jika nama tersebut memiliki arti seperti int tanggal[10].

C. Fungsi

Fungsi adalah suatu bagian dari program yang dirancang untuk melaksanakan tugas tertentu dan letaknya dipisahkan dari program yang menggunakannya. Dengan membuat fungsi, program menjadi terstruktur, sehingga mudah dipahami dan dikembangkan[11].

Dekomposisi kode bisa dilakukan dengan membuat sebuah fungsi, kode program yang panjang dipisahkan dengan sebuah fungsi berdasarkan jenisnya. Sehingga kode yang ditulis menjadi lebih sederhana dan mudah dipahami[12].

D. SonarLint

SonarLint adalah ekstensi IDE yang membantu mendeteksi dan memperbaiki masalah kualitas saat menulis kode program. Seperti pemeriksa ejaan, SonarLint menemukan kekurangan yang dapat diperbaiki pada sebuah kode.

SonarLint dapat membantu meningkatkan kualitas dari sebuah kode. SonarLint dapat membantu dalam menemukan *bugs* dan *issue.* SonarLint juga memberikan *feedback* tentang kode yang dituliskan[13].

E. PHP CS Fixer

PHP CS Fixer adalah *tools* untuk memperbaiki sebuah kode dengan mengikuti standar pengkodean bahasa pemrograman PHP yang mengikuti standar yang didefinisikan PSR-1, PSR-2 atau bisa mengikuti standar tim dengan melakukan konfigurasi manual[14].

Dengan PHP CS Fixer dapat membuat kode lebih mudah dibaca. Hal-hal yang menentukan kode lebih mudah dibaca dengan memperhatikan, *indentation, line break* dan *whitespace* agar kode terlihat rapi. PHP CS Fixer dapat memperbaiki hal tersebut[15].

III. METODOLOGI

Penelitian dimulai dengan mempelajari konsep *clean code* dan penerapannya terhadap proyek. Mempelajari konsep *clean code* dengan membaca artikel tentang apa itu *clean code* dan penerapannya pada *project*. Tahap selanjutnya yaitu mengikuti *guidelines* pada perusahaan. *Guidelines* berisi aturan dan ketentuan dalam penulisan *variable* dan menulis kode yang baik. Setelah memahami konsep *clean code* dan mengikuti *guidelines* tahap selanjutnya yaitu mengimplementasikan aturan dan konsep tersebut ke dalam *project*, dengan cara menuliskan kode sesuai dengan konsep *clean code* dan aturan dari perusahaan.

A. Penamaan

Tahap ini merupakan tahapan pertama dalam mengimplementasikan *clean code* pada *project.* Menentukan nama pada setiap *variable* yang akan digunakan dalam menuliskan kode. Aturan penamaan mengikuti aturan yang sudah dibuat pada perusahaan. Pada tahap ini memastikan anggota tim memahami tujuan dari kode tersebut.

B. Duplikasi Kode

Tahap selanjutnya, menghindari agar terjadinya duplikasi setiap kode. Kode yang terduplikasi membuat baris kode menjadi lebih banyak, susah untuk dibaca dan dimodifikasi.

C. Kualitas Kode

Proggramer yang baik yaitu yang dapat menuliskan kode yang dapat dibaca dan mudah dipahami. Pada tahap ini berfungsi untuk memastikan kualitas kode yang dituliskan baik. Pengujian dilakukan dengan menggunakan *tools* SonarLint dan PHP CS FIXER.

IV. HASIL DAN PEMBAHASAN

A. Penamaan

Nama sangat penting dalam menuliskan sebuah kode, nama *variable*, fungsi atau kelas yang bermakna dapat menjelaskan apa yang dimaksud dalam baris kode. Nama dengan konteks dan makna yang jelas dapat memberikan arti pada keseluruhan baris kode. Tabel 1 adalah penulisan nama yang ditetapkan oleh perusahaan[16].

TABLE I. ATURAN PENAMAAN

What	**Penulisan**	**Contoh**
Variable	*camelCase*	$userId
Class Property	*camelCase*	private $accessToken
Class Method	*camelCase*	$postRepository->featuredArticle()
Model	*StudlylCase*	User UserProfile

Dalam penulisan *variable,* fungsi dan *method* menggunakan penulisan *camelCase* sedangkan untuk model menggunakan penulisan *StudlyCase* mengikuti aturan perusahaan[16]. Dalam menuliskan nama kelas atau model, pastikan menggunakan kata benda seperti *Package,* Instansi. Untuk penamaan *method* menggunakan kata kerja seperti *getDataCustomer* dan *getDataDukung*.

Gambar 1 menunjukkan penamaan *variable* dengan penulisan *camelCase* dengan 2 kata, kata pertama diawali dengan huruf kecil yang dilanjutkan dengan kata kedua dengan diawali huruf besar tanpa terpisah. Untuk penamaan *variable* dengan 1 kata menggunakan huruf kecil.

Gambar 2 menunjukkan penamaan kelas model dengan penulisan *StudlyCase* dengan 2 kata, kedua kata menggunakan huruf besar tanpa dipisah.

Gambar 3 menunjukkan penaaman kelas *method,* aturan penamaan kelas *method* sama dengan aturan penamaan pada *variable* dengan menggunakan penulisan *camelCase.* Dalam penulisan nama lebih detailnya ada di Gambar 1 , Gambar 2 dan Gambar 3.

Gambar 1. Nama *Variable*

Gambar 2. Nama Fungsi dan Model

Gambar 3. Nama *Method*

B. Duplikasi Kode

Suatu fungsi harus melakukan satu hal, jika suatu fungsi melakukan banyak hal, dapat menyebabkan kebingungan. Setiap pernyataan dalam fungsi tidak boleh lebih dari satu, karena fungsi memiliki tujuan yang jelas[17].

Apabila kode tidak terbungkus dalam suatu *method* atau fungsi, jika terjadi suatu perubahan maka semua kode harus diubah. Dengan adanya fungsi dan *method*, kode menjadi lebih terstruktur, dapat mengurangi kode yang terduplikasi. Kode program yang sama dan dipakai berulang dapat dituliskan sekali secara terpisah dalam bentuk fungsi dan *method*. *Refactor* kode menjadi lebih mudah.

Pada fungsi *create* dan edit terdapat sebuah kode untuk mengambil data, yaitu data *package* dan komitmen, karena data yang diambil sama pada setiap fungsi, maka dibuatlah sebuah *method getPackage* dan *getKomitmen*, dengan menggunakan *method* atau fungsi, kode untuk mengambil data *package* dan komitmen pada fungsi create dan edit tidak perlu ditulis ulang di kedua fungsi tersebut, tinggal memanggil method yang telah dibuat dapat mengurangi duplikasi kode dan kode terlihat lebih sederhana.

Gambar 4 menunjukkan fungsi *create* dan edit, di dalam fungsi tersebut terdapat sebuah kode untuk memanggil sebuah method yang sama.

Gambar 4. Fungsi *Create* dan Edit

Gambar 5 menunjukkan kode yang ada pada setaip method untuk mengambil sebuah data.

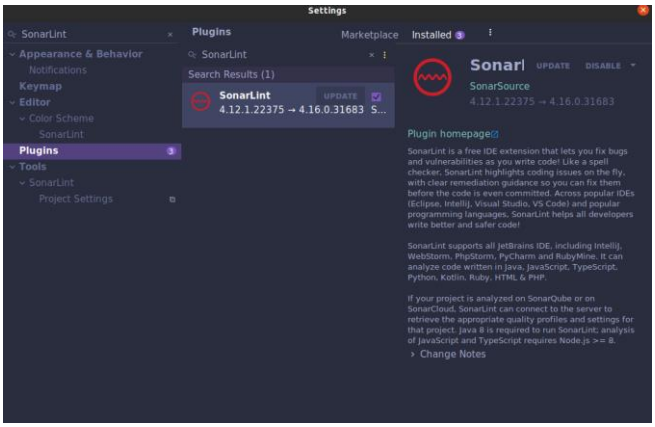
Gambar 5. *Method* getPackage dan getKomitmen

C. Kualitas Kode

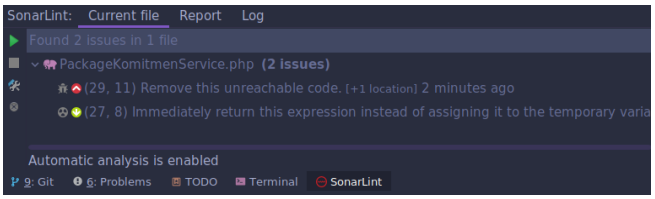
Kualitas kode mendefinisikan kualitas seorang *programmer* dalam menuliskan kode. Untuk meningkatkan kualitas kode terdapat *tool* SonarLint yang dipasang di *IDE* dan PHP CS Fixer yang dapat dipasang pada *project* aplikasi.

Tool SonarLint dapat dipasang di *IDE* seperti pada Gambar 6 yang menunjukkan bahwa SonarLint sudah terpasang pada *IDE*. Dengan menjalankan SonarLint untuk melakukan pengujian, Sonarlint akan menampilkan *error* pada kode jika terdapat sebuah kesalahan.

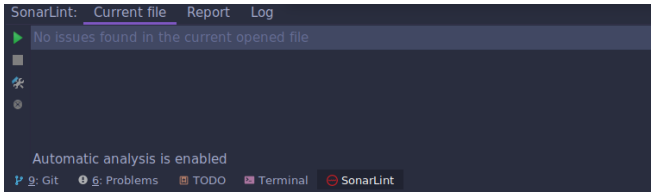
Gambar 7 menampilkan sebuah kesalahan yang terdeteksi dengan SonarLint. SonarLint akan menunjukkan pesan kesalahan. Pada Gambar 7 dijelaskan bahwa ada sebuah kode yang tidak terpakai dan tidak dapat dijalankan, sehingga kode dapat dihapus atau diperbaiki. Pada Gambar 8 tidak menampilkan kesalahan kode yang terdeteksi pada SonarLint.



Gambar 6. SonarLint



Gambar 7. Contoh Kesalahan Terdeteksi



Gambar 8. Contoh Tidak Terdeteksi Kesalahan

V. KESIMPULAN

Berdasarkan hasil pembahasan diatas, dapat disimpulkan bawah dengan adanya konsep *clean code*, *programmer* dapat saling memahami maksud dan tujuan dari kode yang dituliskan, karena kode yang dituliskan mudah dibaca dan dipahami. Kode yang mudah dibaca dan dipahami dapat memudahkan pekerjaan *programmer* dalam kolaborasi antar tim untuk melanjutkan kode jika ada perubahan yang terjadi. Penulisan sebuah nama *variable* sangat membantu dalam mengikuti konsep *clean code*.

Dengan memanfaatkan *tool* SonarLint, dapat menghemat waktu dalam melakukan perbaikan kode, karena mencari kesalahan sebuah kode dilakukan secara otomatis dan PHP CS Fixer sangat membantu dalam memperbaiki *coding style* karena perbaikan dilakukan secara otomatis.

Adapun saran guna meningkatkan hasil penelitan selanjutnya, yaitu diperlukan penelitian lebih lanjut untuk mengetahui impelementasi *clean code* pada *design pattern* dan penelitian lebih lanjut untuk mengetahui peran *tool* SonarLint dalam konsep *Clean Code*.

REFERENCES

[1] A. Supriyatna and M. A. S. Nugroho, “Sistem Informasi Forum Diskusi Programmer Berbasis Web Menggunakan Rapid Application Development,” Teknika, vol. 7, no. 2, pp. 138–147, 2018, doi: 10.34148/teknika.v7i2.132.

[2] L. W. Dietz, “Teaching Clean Code,” pp. 40–43, 2017.

[3] S. Sunny, “What is Bad Code? How to Write Clean Code?,” Noteworthly-The Journal Blog, 2019. https://blog.usejournal.com/what-is-bad-code-how-to-write-clean-code-a9b7b539ad8.

[4] Wellcode.io Team, “WSINSIGHT,” 2020. https://insight.wellcode.io/clean-code.

[5] Vinashaw, “Apa yang dimaksud dengan kode program,” dictio, 2018. https://www.dictio.id/t/apa-yang-dimaksud-dengan-source-code-atau-kode-program/15085.

[6] Z. Muhammad, “Apa itu Clean Code dan kenapa begitu penting?,” Medium, 2020. https://medium.com/itmi-engineering/apa-itu-clean-code-dan-kenapa-begitun-penting-part-1-3e0fce6984c.

[7] L. Natalia, “Clean Code,” Binus Univeristy, 2014. https://sis.binus.ac.id/2014/04/12/clean-code/.

[8] S. Didik tri, “5 Cara Mudah Menerapkan Clean Code,” DOT Blog, 2018. https://blog.dot.co.id/5-cara-mudah-menerapkan-clean-code-b2e0ec1b860e.

[9] G. Beniamini, S. Gingichashvili, A. K. Orbach, and D. G. Feitelson, “Meaningful Identifier Names: The Case of Single-Letter Variables,” 2017.

[10] A. Budiarti, “Bab 2 landasan teori,” Apl. dan Anal. Lit. Fasilkom UI, pp. 4–25, 2006.

[11] V. Bab, “Yaitu Untuk Menampilkan Informasi Atau Data Ke Layar.,” pp. 64–91.

[12] N. Andi Taru, “Pentingnya Menerapkan Clean Code,” Gamelab.id, 2019. https://www.gamelab.id/news/153-perhatian-programmer-inilah-pentingnya-menerapkan-clean-code.

[13] W. Dikih Arif, “Meningkatkan ‘Code Quality’ dengan Plugin SonarLint di IntelliJ IDEA,” Javan Cipta Solusi, 2020. https://blog.javan.co.id/meningkatkan-code-quality-dengan-plugin-sonarlint-di-intellij-idea-36705b6cd8fa.

[14] D. Ruminski, “Php CS FIXER,” 2021. https://github.com/FriendsOfPHP/PHP-CS-Fixer.

[15] Dinar, “SEBUAH SENI MENERAPKAN ‘CLEAN CODE,’” Tekno Sejahtera, 2020. https://teknosejahtera.com/sebuah-seni-menerapkan-clean-code/.

[16] Javan, “Laravolt.” https://laravolt.dev/docs/v4/guidelines/naming-things/.

[17] H. G. Koller, “Effects of Clean Code on Understandability,” pp. 1–86, 2016.