Konsistensi Kode pada Bahasa Pemrograman JavaScript Menggunakan Linter pada Continuous Integration Pipeline

Muhammad Paridudin Zia[1] , Dana Sulistyo Kusumo[2] , Donni Richasdy[3]

[1,2,3] Fakultas Informatika, Universitas Telkom, Bandung
[1]muhfaridzia@student.telkomuniversity.ac.id, [2]danakusumo@telkomuniversity.ac.id,
[3]donnir@telkomuniversity.ac.id

Abstrak

Salah satu masalah pada saat proses pengembangan perangkat lunak adalah tidak konsisten nya kode dari bahasa pemrograman yang digunakan, yang mana akan berakibat pada sulitnya kode tersebut dibaca, dipahami dan dimodifikasi di kemudian hari. Maka dari itu dibutuhkan sebuah aturan gay a pengkodean pada bah asa pemrograman yang digunakan. Dengan menggunakan aturan gaya pengkodea n yang sudah ditetapkan dapat memastikan kode program tetap konsisten. Untuk mengatasi masalah tid ak konsisten nya kode program menggunakan linter yang mana bertujuan untuk meningkatkan kualitas pengidentifikasian, pemformatan kode, dan gaya pengkodean yang sesuai dengan style guide JavaScrip t dari Airbnb Engineering. Pada bahasa pemrograman JavaScript, linter yang digunakan adalah ESLint. Dimana memanfaatkan custom ESLint rules sebagai alat untuk memastikan kode JavaScript yang ditulis sesuai dengan aturan yang sudah ditetapkan dan kemudian diimplementasikan pada continuous integration pipeline untuk pengujian kode JavaScript nya. Hasil pengujian menunjukan custom ESLint rules yang dii mplementasikan pada continuous integration pipeline dapat menemukan dan mendeteksi kode program Ja vaScript yang tidak konsisten mengikuti aturan style guide dari Airbnb Engineering. Serta dapat membantu pengembang untuk secara mudah dan cepat melakukan review terhadap kode program yang ditulis oleh pengembang yang lainnya.

Kata kunci : JavaScript, Linter, ESLint, Konsistensi Kode, Continuous Integration

Abstract

One of the problems during the software development process is the inconsistency of the code in the programming language used, which results in difficulty reading, understanding and modifying the code. Therefore we need a coding style rule in the programming language used. Using predefined coding style rules ensures that program code remains consistent. To solve the problem of code inconsistency using a linter which aims to improve the quality of identification, code formatting, and coding style in accordance with the JavaScript style guide from Airbnb Engineering. In the JavaScript programming language, the linter used is ESLint. Which uses custom ESLint rules as a tool to ensure JavaScript code is written according to predefined rules and is then implemented in the continuous integration pipeline for testing JavaScript code. The test results show that the custom ESLint rules that are implemented in the continuous integration pipeline can find and detect JavaScript code that is inconsistent with the Airbnb Engineering style guide. And can help developers to easily and quickly review program code written by other developers.

Keywords: JavaScript, Linter, ESLint, Code Consistency, Continuous Integration

1. Pendahuluan

1.1 Latar Belakang

Pada saat proses pengembangan perangkat lunak salah satu masalah yang sering ditemui oleh pengembang adalah tidak konsisten nya kode dari bahasa pemrograman yang digunakan [1], dimana kode yang tidak konsisten tersebut berakibat kepada sulitnya kode tersebut dibaca, dipahami dan dimodifikasi di kemudian hari [2]. Continuous Integration (CI) adalah praktik dalam industri pengembangan perangkat lunak yang memungkinkan organisasi untuk secara rutin dan andal merilis fitur dan produk baru secara cepat [3]. Salah satu upaya untuk mengatasi masalah tidak konsisten nya kode program pada saat proses pengembangan perangkat lunak adalah dengan memanfaatkan Continuous Integration dan Linter [4] agar kode program mudah dipahami, dimodifikasi serta dipelihara kedepannya oleh anggota tim [5].

Dengan memanfaatkan continuous integration maka pengembang dapat melakukan pengecekan kode pada shared repository secara otomatis dan cepat [3], sedangkan Linter merupakan sebuah static analysis tools yang digunakan untuk memeriksa kemungkinan kode program yang tidak efisien dan masalah gaya pengkodean [6]. Linter dapat membantu pengembang untuk menemukan bug, meningkatkan keterbacaan kode, dan memastikan kode tetap konsisten di seluruh proyek [7]. Dengan menggunakan aturan gaya pengkodean yang sudah di tetapkan maka akan menjaga kode tetap konsisten [8]. Code Convention pada JavaScript yang banyak digunakan untuk keperluan standarisasi kode adalah style guide dari Airbnb Engineering yang digunakan bersamaan dengan ESLint [9]. ESLint dirancang untuk menjadi salah satu linter yang sangat fleksibel, di mana mudah disesuaikan dan digunakan.

Pada saat pengembangan aplikasi khususnya pada bahasa pemrograman JavaScript semakin banyak kode yang ditulis maka akan meningkatkan kompleksitas dari aplikasi yang sedang dikembangkan yang mana mengakibatkan aplikasi semakin susah untuk di uji secara manual untuk mendeteksi kode yang tidak konsisten [4]. Maka dari itu dibutuhkan sebuah cara dan aturan gaya penulisan kode program untuk memfasilitasi pengembang untuk secara mudah dan cepat melakukan pengecekan konsistensi kode. Pada penelitian ini penulis menggunakan linter berbasis ESLint sebagai alat untuk menjaga kode program JavaScript tetap konsisten, style guide dari Airbnb Engineering sebagai rujukan rules dan continuous integration pipeline untuk melakukan pengecekan kode program pada repository.

1.2 Topik dan Batasannya

Topik yang dibahas adalah bagaimana menjaga konsistensi kode JavaScript sesuai rules dari Airbnb Engineering menggunakan linter berbasis ESLint yang diimplementasikan pada continuous integration pipeline.

Batasan dari penelitian yang dilakukan diantaranya adalah sebagai berikut:

1. Tools untuk kebutuhan linter nya menggunakan ESLint.
2. Custom ESLint rules yang diimplementasikan hanya 12 aturan.
3. Bahasa pemrograman yang digunakan untuk pengujian konsistensi kodenya adalah JavaScript dengan mengikuti style guide dari Airbnb Engineering.

1.3 Tujuan

Tujuan yang ingin dicapai dari pembuatan tugas akhir ini adalah dapat mengimplementasikan dan menjaga konsistensi kode JavaScript menggunakan linter berbasis ESLint untuk mendeteksi seberapa banyak kode yang tidak menerapkan aturan style guide dari Airbnb Engineering pada library open source berbasis JavaScript.

1.4 Organisasi Tulisan

1. Bagian 2 menjelaskan mengenai studi terkait pengerjaan tugas akhir.
2. Bagian 3 menjelaskan mengenai rancangan sistem kode konsistensi.
3. Bagian 4 berisi hasil pengujian dan analisis hasil pengujian.
4. Bagian 5 berisi kesimpulan mengenai tugas akhir.
5. Bagian terakhir berisi daftar pustaka yang digunakan pada tugas akhir.

2. Studi Terkait

2.1 Konsistensi Kode

Konsistensi Kode berarti mengikuti gaya penulisan kode yang telah disepakati untuk digunakan di mana pun kita menulis kode. Membaca kode program bisa menjadi sebuah tantangan. Seringkali pengembang akan menggunakan fungsi yang tidak Anda kenal atau dengan cara yang belum pernah Anda lihat dan gunakan sebelumnya. Jika gaya penulisan kode dari pengembang lain kurang baik maka kode nya mungkin akan sulit dibaca dan dipahami [2]. Bergantung pada aturan konsistensi yang disediakan oleh pengembang. Aturan-aturan ini mendefinisikan invarian dari model desain dan kode sumber yang mana diharuskan dipenuhi agar kode tetap konsisten [8]. Kode

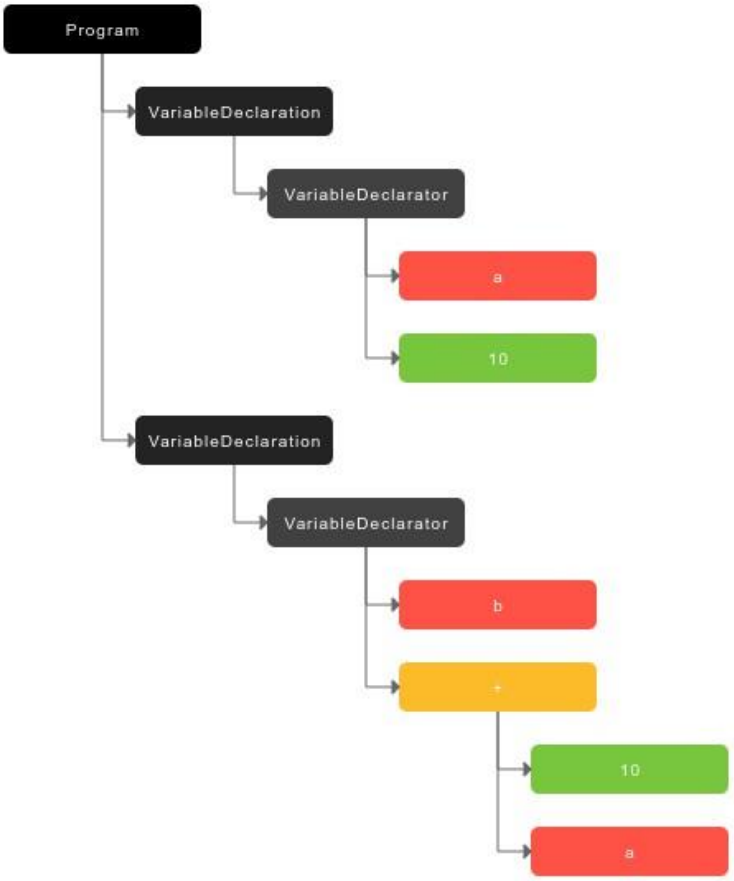
yang konsisten memudahkan orang untuk memahami cara kerja suatu program. Saat membaca kode yang konsisten, seseorang secara tidak sadar membentuk sejumlah asumsi dan harapan tentang cara kerja kode tersebut, sehingga lebih mudah dan aman untuk melakukan modifikasi. Kode yang terlihat sama di dua tempat maka harus bekerja dengan cara yang sama [10].

2.2 Linter

Linter adalah static analysis tools yang digunakan untuk memeriksa kemungkinan kode yang tidak efisien dan masalah gaya pengkodean dimana pattern matching sederhana digunakan dalam banyak static analysis tools [6]. Static analysis tools membantu pengembang untuk menemukan bug, meningkatkan keterbacaan kode, dan memastikan gaya yang konsisten di seluruh proyek [7]. Linter seperti Pylint, ESLint digunakan untuk memeriksa kode dan memberikan peringatan serta rekomendasi kepada pengembang perangkat lunak. Rekomendasi ini dimaksudkan untuk membantu membuat kode yang lebih mudah dibaca oleh manusia, dan untuk mencegah masuknya bug melalui penggunaan idiom yang bermasalah [11].

2.3 Abstract Syntax Tree (AST)

Compier mengubah kode program menjadi representasi yang setara yang disebut dengan Abstr Tak Syntax Tree (AST). ASadalah format tree terstruktur dari kode bahasa pemrograman yang berisi node berbeda dariyang mewakili konstruksi bahasa program. AST membentuk hasil akhir berupa parser setelah evaluasi terben ang tuk dari aturan grammar y dibuat. Syntax analyzer membuat AST menggunakan pendekatan bottom-up. Pend ekatan ini melibatkan p embuatan leaves atau daun dari tree dan kemudian simpul yang dibuat dihubungkan untuk membentuk sebuah str uctured tree [12]. Berikut contoh visualisasi AST dari kode JavaScript let a = 10; let b = 10 + a.



Gambar 1. Visualisasi kode JavaScript menjadi AST

2.4 ESLint

ESLint adalah static analysis tool untuk JavaScript yang ditulis menggunakan Node.js [13]. ESLint dirancang untuk menjadi linter yang sangat fleksibel, di mana ia mudah disesuaikan. ESLint menyediakan sejumlah aturan

dasar yang mana dikelompokkan dalam beberapa kategori berikut: Possible Errors, Best Practices, Strict Mode, Variables, Node.js CommonJS, Stylistic Issues and ECMAScript 6 [14].

2.5 JavaScript Style Guide Airbnb Engineering

Style Guide Airbnb Engineering merupakan sekumpulan aturan penulisan kode untuk bahasa pemrograman JavaScript berbasis open source yang dibuat oleh pengembang dari perusahaan Airbnb dan dibuat tersedia untuk umum [15].

2.6 Continuous Integration

Continuous Integration adalah praktik pengembangan kolaboratif di mana para pengembang seringkali atau setidaknya setiap hari, mengintegrasikan pekerjaan mereka ke dalam shared repository, setelah setiap integrasi selesai dilakukan maka proses build otomatis terjadi [3]. Dengan continuous integration memungki nkan pengem- bang secara cepat mendapatkan feedback ketika mereka selesai melakukan push ke repository uta ma, ini artinya proses pengembangan aplikasi semakin cepat karena proses report nya dilakukan secara otomatis o leh sistem.

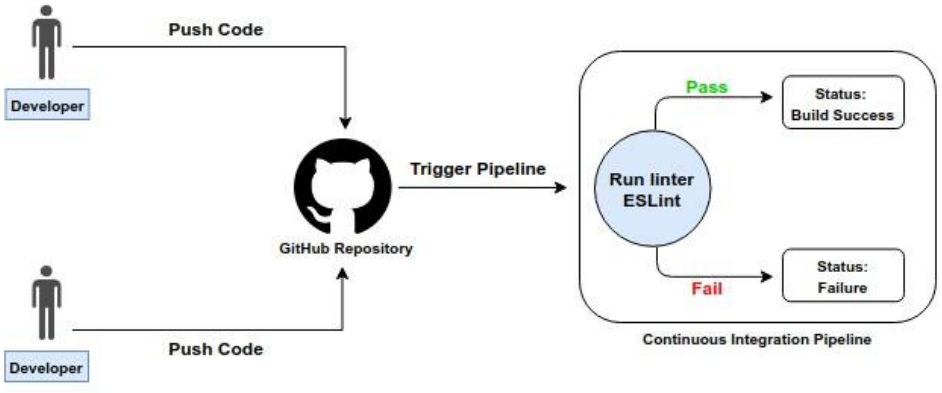
2.7 JavaScript

JavaScript adalah bahasa pemrograman berorientasi objek yang didasarkan pada model objek b erbasis prototi- pe. Bahasa ini terkenal karena penggunaannya sebagai bahasa skrip di web. JavaScript merupakan building block untuk HTML yang Dinamis (DHTML), kumpulan teknologi yang disertakan di hampir semua pera mban web un- tuk mendukung pembuatan situs web animasi yang interaktif [16]. JavaScript sampai saat ini men jadi salah satu bahasa pemrograman yang paling banyak digunakan oleh pengembang di seluruh dunia.

3. Sistem Kode Konsistensi

3.1 Perancangan Sistem

Sistem yang dibangun bertujuan untuk mengimplementasikan kode konsistensi pada bahasa pe mrograman JavaScript dengan menggunakan linter ESLint. Setelah custom ESLint rules selesai dibuat maka dil ak- ukan pengujian terhadap kode program JavaScript yang pada penelitian ini data uji nya menggunakan dua libr ary JavaScript berbasis open source untuk melihat seberapa banyak kode program yang sudah menerapkan atur les an dari custom ESLint rule yang dibuat berdasarkan style guide dari Airbnb Engineering. Pemilihan ESLint se rta style guide Airbnb didasarkan pada kemudahan dan popularitas nya di komunitas JavaScript [9].



Gambar 2. Alur Kerja Sistem Kode Konsistensi

Gambar 2 menjelaskan bagaimana alur kerja dari sistem yang dibangun, dimana ketika pengembang melakukan commit dan push kode ke GitHub repository maka secara otomatis continuous integration dijalankan dan akan memicu pipeline untuk menjalankan tugas yang diberikan, yang dalam hal ini adalah menjalankan linter ESLint untuk melakukan pengujian terhadap kode program JavaScript sesuai dengan style guide dari Airbnb Engineering. Jika kode yang diuji berhasil lolos dalam pengujian maka kode tersebut akan masuk ke main branch, namun jika pada saat pengujian kode program belum sesuai dengan rules yang sudah ditentukan, linter akan memberikan pesan error yang sesuai kepada pengembang untuk dilakukan perbaikan.

3.2 Proses Pembuatan Custom Rules

Berdasarkan kajian pustaka yang dilakukan, penulis tidak menemukan tahapan yang membahas detail dalam pembuatan custom rules. Namun untuk proses pembuatan custom rules nya mengacu pada editor AST Explorer.

Gambar 3. Proses Pembuatan Custom Rules ESLint

Tahap awal dari proses pembuatan custom rules pada editor AST Explorer adalah dengan menentukan input code yang akan diujikan. Berikutnya memilih jenis parser dan transformer yang digunakan untuk melakukan parsing dan transformasi kode program ke AST, untuk jenis parser nya sendiri dapat memilih babel-eslint kemudian untuk transformer code nya menggunakan ESLint v4. Setelah itu editor secara otomatis akan menghasilkan Abstract Syntax Tree (AST) dari masukan kode program JavaScript yang diberikan. Dari AST ini kemudian pengembang dapat menentukan node mana saja yang diperlukan untuk kebutuhan pengecekan konsistensi kode programnya. Setelah pengembang selesai menulis kode program untuk transformer code nya, maka rules tersebut sudah dapat digunakan untuk melakukan pengujian terhadap kode JavaScript yang sesuai.

3.3 Implementasi ESLint Rules Airbnb

Custom ESLint Rules yang diimplementasikan pada penelitian ini adalah berjumlah 12 rules atau sebanyak 8.219% dari total keseluruhan kategori pada rules Airbnb Engineering. Dimana untuk pemilahan rules diambil berdasarkan rules yang sering digunakan oleh pengembang. Pada penelitian ini penulis memilih 4 kategori rules yaitu Stylistic Issues, Variables, Best Practice dan ECMAScript 6 [9].

Tabel 1. Implementasi Rules

No	Nama	Kategori	Deskripsi
1	ta-no-assign	References (ES6)	Jangan melakukan reassigning variabel yang dibuat dengan const
2	ta-no-var	References (ES6)	Gunakan let atau const daripada var untuk pembuatan variabel
3	ta-no-chained-var	Variables (Stylistic Issues)	Jangan menggunakan chained ekspresi pada saat assignment nilai
4	ta-no-undeclared-var	Variables (Variable)	Jangan menggunakan variabel yang tidak dideklarasikan, kecuai ada di dalam ‘/*global */
5	ta-no-dup-class	Classes (ES6)	Dilarang mendeklarasikan class members secara duplikat
6	ta-no-unnecesary-constructor	Classes (ES6)	Dilarang menggunakan construtor yang tidak diperlukan
7	ta-dup-import	ES-Modules (ES6)	Dilarang mendeklarasikan module imports secara duplikat
8	ta-no-new-object	Objects (Stylistic Issues)	Gunakan objek literal daripada objek constructors
9	ta-arrow-spacing	Arrow Function (ES6)	Menerapkan spasi yang konsisten sebelum dan sesudah arrow
10	ta-use-brace-style	Block Statement (Stylistic Issues)	Gunakan brace style yang konsisten untuk block statement
11	ta-use-rest-parameter	Function (ES6)	Gunakan rest parameter daripada arguments
12	ta-equality-operator	Operator and Comparison (Best Practices)	Gunakan operator === dan !== untuk melakukan perbandingan

4. Evaluasi

Pengujian pertama difokuskan untuk menguji konsistensi kode dari library JavaScript berbasis open source dan pengujian kedua berfokus pada pengujian dari sisi kolaborasi antar pengembang untuk mengetahui apakah linter yang diimplementasikan pada continuous integration dapat mendeteksi kode yang tidak konsisten mengikuti aturan, kemudian untuk mengetahui tingkat kemudahan code review kode program nya, apakah lebih mudah jika menggunakan linter atau tidak. Custom ESLint Rules dan Continuous Integration Pipeline dikatakan berhasil melakukan tugasnya apabila dapat melakukan pendeteksian dan memberikan pesan error terhadap kode program yang tidak sesuai dengan style guide dari Airbnb Engineering

4.1 Skenario Pengujian

4.1.1 Pengujian 1 Library JavaScript

Pada tahap ini dilakukan pengujian terhadap library JavaScript bernama Pixi-live 2d yang merupakan library JavaScript berbasis open source untuk menampilkan model 2D secara langsung sebagai sprite di browser menggunakan custom ESLint rules yang dijalankan pada continuous integration pipeline untuk menguji konsistensi kode program JavaScript nya.

4.1.2 Pengujian 2 Kolaborasi antar Pengembang

Pengujian kedua merupakan pengujian dari sisi kolaborasi antar pengembang, dimana pada pengujian ini dua pengembang dilibatkan, pengembang pertama dan kedua akan bergantian untuk menjadi reviewer dari kode program JavaScript yang ditulis untuk mengetahui apakah pemakaian linter dapat memudahkan code review dan juga untuk memastikan custom ESLint rules dapat mendeteksi kode program yang tidak mengikuti aturan style guide dari Airbnb Engineering.

4.2 Analisis Hasil Pengujian

4.2.1 Analisis Hasil Pengujian 1

Dari hasil pengujian yang dilakukan pada library Pixi-live 2d sebagaimana yang ditunjukan pada Tabel 2 terdapat total 878 error dari 12 rules yang diujikan. Dimana 260 (29.6%) error ditemukan pada rules ta-no-var, 324 (36.9%) error pada rules ta-no-undeclared-var, 203 (23.1%) error pada rules ta-use-brace-style, 91 (10.3%) error pada rules ta-equality-operator dan untuk rules lainnya sudah sesuai dengan style guide dari Airbnb Engineering.

Tabel 2. Hasil Pengujian pada Library JavaScript Pixi-live 2d

No	Nama	Kategori	Jumlah Error	Persentase
1	ta-no-assign	References (ES6)	0	0
2	ta-no-var	References (ES6)	260	29.6%
3	ta-no-chained-var	Variables (Stylistic Issues)	0	0
4	ta-no-undeclared-var	Variables (Variable)	324	36.9%
5	ta-no-dup-class	Classes (ES6)	0	0
6	ta-no-unnecesary-constructor	Classes (ES6)	0	0
7	ta-dup-import	ES-Modules (ES6)	0	0
8	ta-no-new-object	Objects (Stylistic Issues)	0	0
9	ta-arrow-spacing	Arrow Function (ES6)	0	0
10	ta-use-brace-style	Block Statement (Stylistic Issues)	203	23.1%
11	ta-use-rest-parameter	Function (ES6)	0	0
12	ta-equality-operator	Operator and Comparison (Best Practices)	91	10.3%

Setelah melakukan pengujian pada library open source Pixi-live 2d menggunakan custom ESLint rules pada continuous integration pipeline masih ditemukan error sebagaimana yang ditunjukan pada Tabel 2 yang disebabkan oleh pengembang yang menulis kode program tidak mengikuti aturan penulisan kode JavaScript sesuai style guide dari Airbnb Engineering. Ketika pengembang melakukan commit dan push kode program ke Git repository, secara otomatis custom ESLint rules yang sudah terpasang pada continuous integration pipeline dijalankan dan melakukan pengujian untuk mendeteksi apakah ada kode program yang tidak sesuai dengan rules dari JavaScript Airbnb Engineering.

Dengan menggunakan linter maka pengembang dapat merancang rules nya sendiri agar pengembang lain yang menulis kode JavaScript mengikuti aturan penulisan yang sudah dibuat, kemudian dengan adanya continuous integration pipeline yang sudah terpasang linter didalamnya dapat memastikan kode JavaScript yang di push oleh penggembang ke remote Git repository tetap konsisten dikarenakan jika pipeline tidak pass maka secara otomatis continuous integration akan memberikan pesan error dan kode tidak dapat di merge ke main branch karena masih terdapat kesalahan pada kode program. Untuk mendapatkan hasil yang konsisten untuk keseluruhan kode program maka pengembang harus mengikuti semua rules yang sudah ditetapkan yang dalam hal ini adalah rules dari Airbnb Engineering.

4.2.2 Analisis Hasil Pengujian 2

Pada pengujian ini melibatkan dua orang responden sebagai sampel pengembang, yang mana bertugas untuk menulis kode dan melakukan review terhadap kode program JavaScript yang sudah di push ke remote repository.

Tabel 3. Background Pengembang

Background	Experience	Skill
JavaScript Developer	1 Tahun	Intermediate
Informatics Student	6 Bulan	Novice

Dari hasil pengujian yang telah dilakukan oleh dua orang pengembang, yang mana berperan sebagai penulis kode program dan melakukan review terhadap kode JavaScript nya. Didapatkan respons sebagaimana yang ditunjukan pada Tabel 4 dan Tabel 5. Dimana berdasarkan dari dua respons tersebut penggunaan linter dapat memudahkan dan mempercepat proses review kode yang ditulis oleh pengembang lain.

Tabel 4. Hasil Respons Pengembang 1

Pertanyaan	Ya	Tidak	Alasan
Linter memudahkan melakukan code review terhadap kode JavaScript yang ditulis oleh pengembang lain?	Ya		Karena menggunakan linter dapat memangkas waktu untuk melakukan review terhadap kode javascript yang ditulis. Sehingga pengembang dapat lebih fokus untuk me-review logika bisnis aplikasi.

Tabel 5. Hasil Respons Pengembang 2

Pertanyaan	Ya	Tidak	Alasan
Linter memudahkan melakukan code review terhadap kode JavaScript yang ditulis oleh pengembang lain?	Ya		Tidak perlu mengeluarkan usaha yang lebih untuk melakukan review kode dari pengembang lain karena bisa dilakukan di level continuous integration.

5. Kesimpulan

Pada penelitian ini bertujuan untuk menjaga konsistensi kode program yang ditulis menggunakan bahasa pemrograman JavaScript, dimana dari hasil pengujian yang sudah dilakukan, custom ESLint rules yang telah dipasang pada continuous integration pipeline dapat menemukan dan mendeteksi kode program yang tidak konsisten mengikuti aturan dari style guide Airbnb Engineering serta dapat membantu pengembang untuk secara mudah dan cepat melakukan review terhadap kode program yang ditulis oleh pengembang yang lainnya. Sehingga dapat disimpulkan dengan menggunakan linter yang dalam hal ini adalah ESLint dan dikombinasikan dengan continuous

integration pipeline dapat membantu menjaga kualitas kode program tetap konsisten mengikuti aturan yang sudah disepakati dan dapat memangkas waktu pengembang dalam melakukan code review.

Untuk penelitian selanjutnya bisa mengimplementasikan lebih banyak rules untuk dapat mendeteksi lebih banyak kode program yang tidak konsisten. Selain itu, dapat mempertimbangkan untuk membandingkan linter berbasis JavaScript yang lainnya dengan ESLint untuk mengetahui mana jenis linter yang lebih efisien dalam pembuatannya, kemudian dapat pula melakukan perbandingan terhadap code convention JavaScript lainnya dengan milik Airbnb Engineering.

Referensi

[1] Anthony CW Finkelstein, Dov Gabbay, Anthony Hunter, Jeff Kramer, and Bashar Nuseibeh. Inconsistency handling in multiperspective specifications. IEEE Transactions on Software Engineering, 20(8):569–578, 1994.

[2] Cons istent Code Style, [Accessed: 11-Feb-2020]. http://dept.stat.lsa.umich.ed u/~jerrick/ cour ses/stat701/notes/codestyle.html.

[3] Rob bert Jongeling, Jan Carlson, Antonio Cicchetti, and Federico Ciccozzi. Continuous integ ration support in m odeling tools. CEUR Workshop Proceedings, 2245:268–276, 2018.

[4] Jose Ivan Vargas. How DevOps and GitLab CI/CD enhance a frontend workflow, 2018 [accessed: 10 Febr uary 2020]. https://about.gitlab.com/blog/2018/08/09/how-devops-and-gitlab-cicd-enhan ce-a-frontend- work flow.

[5] Björn Latte, Soreñ Henning, and Maik Wojcieszak. Clean code: On the use of practices and ols to produce main to tainable code for long-living. 2019.

[6] Matic Potocnik, Uroš Cibej, ˇ and Bostjaň Slivnik. Linter - A tool for finding bugs and al problems in Scala potenti code. Proceedings of the ACM Symposium on Applied Computing, pages 1615–014. 1616, 2

[7] Caitl rder: Building a pro in Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Soderbeï g, and Collin Winter, ng, 1:598–608, 2015. Trico gram analysis ecosystem. Proceedings - International Conference on Software Engineeri

[8] Mar de consistency chec kus Riedl-Ehrenleitner, Andreas Demuth, and Alexander Egyed. Towards model-and-co , 2014.

[9] Kristí king. In 2014 IEEE 38th Annual Computer Software and Applications Conference. IEEE cript linters in pract n Fjolá Tomasd ottiŕ , Mauricio Aniche, and Arie Van Deursen. The adoption

[10] The I of javas ice: A case study on eslint. IEEE Transactions on Software Engineering, 2018. .gnome.org/ prog mportance of Writing Good Code, [Accessed: 11-Feb-2020]. https://developer

[11] Nick ramming-guidelines/stable/writing-good-code.html.en. hecking for ml data s Hynes, D Sculley, and Michael Terry. The data linter: Lightweight, automated sanity c

[12] Sambit Kumar Patra, Binod Kumar Pattanayak, and Bhagabat Puthal. Optimizing ast node for javascript compiler a lightweight interpreter for embedded device. Journal of Computers, 8(2):349, 2013.

[13] Harry Cummings. Learning Node.js for .NET Developers. Packt Publishing Ltd, 2016.

[14] Kristín Fjóla Tómasdóttir, Mauricio Aniche, and Arie van Deursen. Why and how javascript developers use linters. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2017.

[15] Airbnb Engineering Team. Airbnb JavaScript Style Guide, [Accessed: 5-Feb-2021]. https://airbnb.io/ javascript/.

[16] Tommi Mikkonen and Antero Taivalsaari. Using javascript as a real programming language, 2007.