**Nhật ký tìm hiểu ESLint**

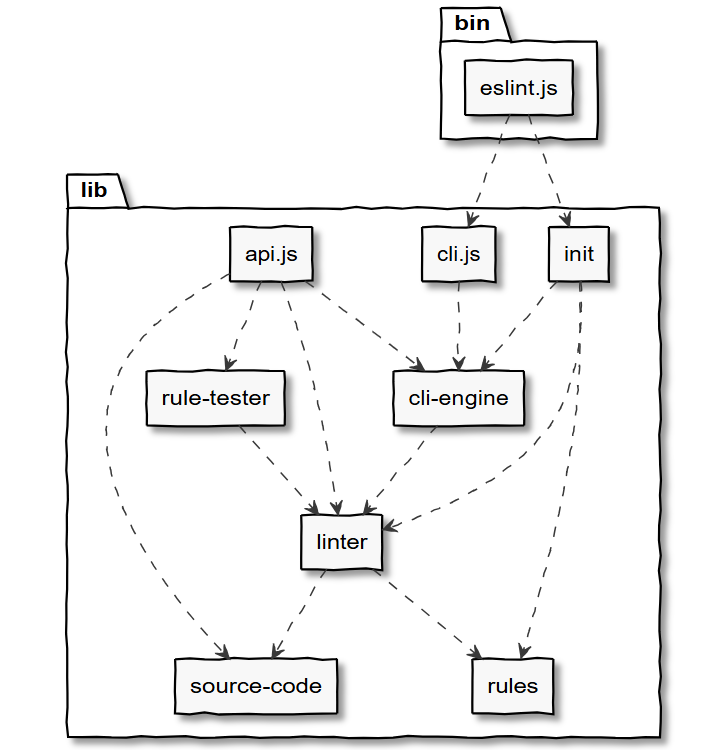
(<https://eslint.org/>)

# 1, Định nghĩa



* Về định nghĩa chung: ESLint là **công cụ phân tích tĩnh** (static analysis tool).
* ESLint được xây dựng chuyên biệt để xử lý code JavaScript, cụ thể:
  + Phát hiện lỗi cú pháp & logic
  + Đảm bảo coding style nhất quán
  + Áp dụng best practices
  + Tối ưu hóa hiệu suất và bảo mật
  + Kiểm tra code đặc thù (React, TypeScript, Node.js,…)
* Về định nghĩa thực tế, ESLint là một thư viện (package) trên npm (Node Package Manager).
* ESLint cung cấp nhiều khả năng làm việc cho dự án:
  + Tích hợp vào IDE
  + Chạy bằng dòng lệnh (CLI)
  + Tích hợp vào Git Hook (trước khi commit lên GitHub)
  + Chạy trong CI/CD pipeline (trước khi deploy)
  + Sử dụng như một module trong mã nguồn (để import)
* Khi mới quen thì dùng CLI trước, sau này tích hợp vào hệ thống sau.

# 2, Các thành phần của ESLint (Architecture)



Nguồn ảnh: <https://eslint.org/docs/latest/contribute/architecture/>

Trong đó cũng nói rất rõ về cấu tạo của eslint rồi, tôi sẽ tóm tắt nó như sau:

Về cơ bản, ESLint gồm 2 thành phần chính: bin (để khởi động module) và lib (chứa những thành phần chính của module để dò lỗi).

## Thư mục bin

* Cũng như các thư viện/module khác thì thư mục bin chứa những file có tác dụng khởi chạy chương trình. Cụ thể ở đây là file **eslint.js**.
* Do chỉ làm nhiệm vụ chuyển tiếp tham số đến **cli.js** (ở dưới), nên không cần nhiều kiểm thử.

## Thư mục lib

* Chứa 8 thành phần: cli.js, init, api.js, rule-tester, cli-engine, linter, source-code, rules.
  + **lib/api.js**:
    - Đây là điểm nhập khi sử dụng **require("eslint")**.
    - Tệp này cung cấp một đối tượng chứa class như Linter, ESLint, RuleTester, và SourceCode, để mình có thể dùng mà không cần gõ code terminal.
  + **lib/cli.js**:
    - Đây là trung tâm của CLI (giao diện dòng lệnh) của ESLint.
    - Đây cũng là phần xử lý việc đọc tệp, duyệt thư mục, nhập và xuất dữ liệu.
  + **lib/cli-engine/:**
    - Đây là module chứa lớp CLIEngine, có nhiệm vụ tìm tệp mã nguồn, tệp cấu hình.
    - CLIEngine đóng vai trò trung gian giữa CLI và Linter, bản thân cli-engine không trực tiếp kiểm tra mã mà chỉ thiết lập môi trường để Linter làm việc.
    - Nó cũng là nơi tải cấu hình cho ESLint (dựa vào file configuration), phân tích cú pháp (parser), mở rộng (nhờ plugin) và bộ định dạng code (formatter).
  + **lib/linter/**:
    - Đây là module chứa class Linter, có chức năng kiểm tra mã dựa trên các tùy chọn cấu hình (được tải và chọn nhờ cli-engine).
    - Nó không thực hiện đọc/ghi file và không tương tác với console.
  + **lib/rule-tester/:**
    - Module này chứa class **RuleTester**, giúp kiểm thử các quy tắc ESLint bằng Mocha (một framework kiểm thử phổ biến trong JavaScript *(lần đầu viết về kiểm thử nên cũng chịu, tự đọc theo* [*link*](https://mochajs.org/) *vậy)*), hoặc framework kiểm thử khác.
    - Hiểu đơn giản hơn, nếu test tay sẽ phải tự nhìn và tự phát hiện những lỗi cơ bản trong cả src code => tốn sức tốn thời gian khi mà chỉ có một tập quy tắc cố định mà cứ phải test đi test lại => rule-tester sinh ra giúp tự động hóa và nhất quán hóa quá trình đó.
  + **lib/source-code/:**
    - Đây là module chứa lớp **SourceCode** (như cái tên, nó đại diện cho mã nguồn đã được phân tích).
    - Cụ thể, chương trình phân tích mã nguồn ra theo dạng AST (Abtract Syntax Tree), ra kết quả là một đối tượng dạng JSON, rồi đối tượng SourceCode sẽ lưu kết quả đó cùng với mã nguồn gốc, token, comment trong code, và vị trí từng phần tử để tiến hành phân tích sau.
  + **lib/rules/**:
    - Chứa các quy tắc tích hợp sẵn để kiểm tra mã nguồn.
    - Các quy tắc của ESLint được viết dưới dạng **các module JavaScript**. Mỗi quy tắc là một file .js và được định nghĩa dưới dạng **một đối tượng JavaScript** chứa các phương thức kiểm tra AST.

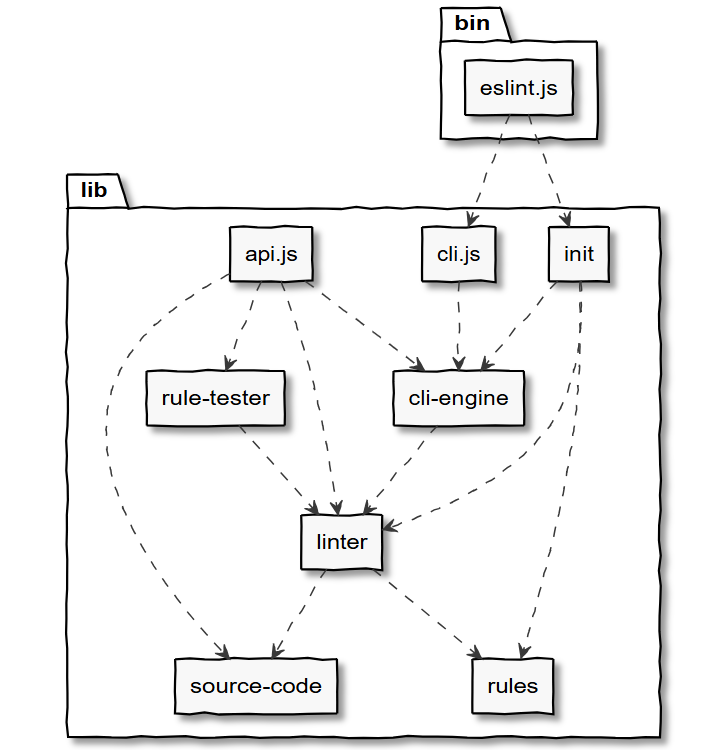


* *Cũng không hẳn là tóm tắt lắm:)) vẫn còn khá confuse khi nhắc tới chức năng của từng phần, nhưng do giới hạn tìm hiểu và thời gian nên đọc cái này cho biết thôi.*

Khi biết được thành phần và chức năng của nó rồi, thì lắp ghép lại sẽ ra cách hoạt động của cả thư viện.

# 3, ESLint hoạt động như thế nào?

Trước hết lưu ý: ESLint là công cụ phân tích TĨNH, nên nó sẽ không chạy code mà chỉ phân tích cấu trúc của code. (theo như chức năng của từng phần nói lúc nãy thì cũng chẳng có cái nào có tác dụng chạy code cả)



(lấy lại hình này một lần nữa vì nó mô tả được luôn thứ tự hoạt động)

## Các bước chi tiết:

1, **Khởi động ESLint (Từ CLI hoặc API)**

**Tham gia**: bin/eslint.js, lib/cli.js

* Khi chạy lệnh eslint, file bin/eslint.js sẽ được gọi.
* eslint.js thực hiện nhiệm vụ chính là chuyển các tham số đầu vào từ CLI đến cli.js trong thư mục lib.
* Quá trình này không thực hiện nhiều xử lý, chỉ chuyển tiếp tham số.

2, **Đọc và Xử lý Tham số CLI**

**Tham gia**: lib/cli.js

* cli.js tiếp nhận tham số từ lệnh (ví dụ: tệp cần kiểm tra, cấu hình, plugin) và bắt đầu thiết lập môi trường cho quá trình kiểm tra.
* Nó sẽ kiểm tra các tệp cấu hình (như .eslintrc), xử lý các tùy chọn, và chuẩn bị dữ liệu cần thiết cho quá trình kiểm tra mã nguồn.

3, **Khởi tạo CLIEngine và Thiết lập Môi Trường**

* **Tham gia**: lib/cli-engine/CLIEngine
* cli.js sẽ khởi tạo đối tượng CLIEngine từ module cli-engine. Đây là lớp trung gian giữa giao diện dòng lệnh (CLI) và quá trình kiểm tra mã.
* CLIEngine sẽ tìm tệp mã nguồn cần kiểm tra và các tệp cấu hình như .eslintrc, thiết lập các tuỳ chọn, và khởi tạo các plugin, parser, formatter.

4, **Tải Cấu Hình và Plugin**

* **Tham gia**: lib/cli-engine/CLIEngine
* CLIEngine sẽ tải các cấu hình (ví dụ: từ tệp .eslintrc), các plugin (như eslint-plugin-react), và các tùy chọn định dạng mã (formatter). Nó thiết lập môi trường cho Linter làm việc.
* Cấu hình sẽ bao gồm các quy tắc (rules), parser, và các plugin cần thiết để kiểm tra mã nguồn.

5, **Kiểm tra Mã Nguồn với Linter**

* **Tham gia**: lib/linter/Linter
* Sau khi môi trường đã được thiết lập, CLIEngine sẽ chuyển sang sử dụng Linter để thực hiện kiểm tra mã nguồn. Linter nhận các tùy chọn cấu hình từ CLIEngine và bắt đầu quá trình phân tích mã.
* Linter sẽ kiểm tra mã theo các quy tắc (rules) đã được thiết lập và báo cáo các lỗi hoặc cảnh báo nếu mã vi phạm các quy tắc đó.

6, **Phân Tích Mã Nguồn Thành Cây Cú Pháp Trừu Tượng (AST)**

* **Tham gia**: lib/source-code/SourceCode
* Linter sẽ sử dụng SourceCode để phân tích mã nguồn. SourceCode chịu trách nhiệm chuyển mã nguồn thành **AST (Abstract Syntax Tree)**, cấu trúc dữ liệu giúp ESLint hiểu được cấu trúc của mã.
* SourceCode lưu trữ thông tin như vị trí các token, comment, và mã gốc để việc kiểm tra trở nên chính xác hơn.

7, **Kiểm tra Quy Tắc (Rules)**

* **Tham gia**: lib/rules/ và các module quy tắc
* Các quy tắc kiểm tra mã được chứa trong thư mục lib/rules/, với mỗi quy tắc là một file JavaScript. Mỗi quy tắc có thể kiểm tra một phần của AST và báo cáo khi phát hiện vấn đề.
* Các quy tắc này có thể yêu cầu các thao tác như kiểm tra tên biến, cách viết dấu chấm phẩy, khoảng trắng, hoặc các quy tắc phức tạp hơn.

8, **Kiểm thử Quy Tắc (Nếu Có)**

* **Tham gia**: lib/rule-tester/RuleTester
* Nếu có yêu cầu kiểm thử quy tắc, RuleTester sẽ được sử dụng để kiểm thử các quy tắc ESLint. Thường quy trình này diễn ra trong giai đoạn phát triển của ESLint hoặc các plugin.
* RuleTester giúp tự động hóa việc kiểm thử các quy tắc, giảm thiểu thời gian và công sức khi phải kiểm thử thủ công.

9, **Báo Cáo Kết Quả**

* **Tham gia**: lib/cli.js, lib/cli-engine/CLIEngine
* Sau khi mã đã được kiểm tra, ESLint sẽ báo cáo kết quả về lỗi hoặc cảnh báo theo định dạng cấu hình (ví dụ: JSON, text, hoặc bảng).
* Kết quả được in ra màn hình hoặc ghi vào file tùy thuộc vào cấu hình, và có thể thấy các vấn đề như lỗi cú pháp, quy tắc vi phạm.

10, **Tự động Sửa Lỗi (Nếu Cần)**

* **Tham gia**: lib/cli.js, lib/linter/Linter
* Nếu chạy ESLint với tuỳ chọn --fix, ESLint sẽ tự động sửa một số lỗi cơ bản như thiếu dấu chấm phẩy, khoảng trắng sai vị trí.

**Bước 5,6,7,8 có thể hơi khó hiểu, nhưng đơn giản hóa nó sẽ như thế này:**

Hãy hình dung ESLint hoạt động giống như một giáo viên chấm bài tập lập trình, trong đó:

* Bước 5: Giáo viên bắt đầu kiểm tra bài (Linter kiểm tra mã nguồn).
* Bước 6: Giáo viên đọc bài và phân tích cấu trúc câu cú (Chuyển code thành AST).
* Bước 7: Giáo viên so sánh với tiêu chí chấm điểm (Rules kiểm tra mã).
* Bước 8: Nếu giáo viên muốn kiểm tra cách chấm điểm có đúng không, họ sẽ tự test lại tiêu chí chấm (RuleTester kiểm thử quy tắc).
* Khá dài và khó nhớ. Nhưng chỉ đọc cho biết chứ cũng không cần nhớ làm gì.

Nếu lười quá thì có thể nhìn hình, hoặc tóm tắt như sau:

## Tóm tắt:

1, **Khởi động ESLint** (bin/eslint.js)

2, **Xử lý Tham số CLI** (lib/cli.js)

3, **Khởi tạo CLIEngine** (lib/cli-engine/CLIEngine)

4, **Tải Cấu Hình và Plugin** (lib/cli-engine/CLIEngine)

5, **Kiểm tra Mã Nguồn với Linter** (lib/linter/Linter)

6, **Phân tích Mã với SourceCode** (lib/source-code/SourceCode)

7, **Báo lỗi**

Cách hoạt động có thể tạm coi là như thế, không cần nắm quá sâu phần này. Với người mới và hướng đến thực hành thì nên nhìn phần này thì hơn:

# 4, Đầu vào và đầu ra của module

Với một module check/sửa lỗi thì có thể đoán ra input là code, output là lỗi và có thể là code (nhưng đẹp hơn). Thật vậy:

* Input

1. Src code: mình chỉ định đường dẫn hoặc tập tin cụ thể để ESLint kiểm tra.
2. Tệp cấu hình: ở file .eslintrc.\*, package.json, hoặc dùng flag (--config). Có thể sửa đổi (người mới thì không nên)
3. Plugin (mình có thể cài đặt thêm từ node\_modules/ (ví dụ: eslint-plugin-react) để ESLint làm việc được với các framework ngoài nữa)
4. Parser:mặc định là espree để đọc cấu trúc code js (có thể thêm như @babel/eslint-parser cho ESNext, JSX)
5. Ruleset: các quy tắc trong tệp cấu hình (rules). Đổi đi khi muốn tùy chỉnh style code.

*Tôi viết thêm space để hiểu rules là “quy tắc trong tệp cấu hình” chứ không phải “tệp cấu hình”*

* Output

1. Danh sách lỗi/cảnh báo (xuất ra terminal hoặc file log).
2. Mã nguồn đã sửa (ghi đè vào file gốc nếu --fix được bật).
3. Báo cáo lỗi (xuất ra terminal hoặc file (theo định dạng chỉ định)).

* Sau này nếu có dùng nhiều thì sẽ để ý kỹ tất cả phần này, nhưng lát nữa demo chỉ chú ý tới phần 1 của mỗi bên thôi.

# 5, Các loại lỗi có thể xuất hiện khi track bằng ESLint

Nói là “các loại lỗi có thể hơi quá, vì có nhiều và có cả những lỗi ít gặp.

Có thể đọc tất cả các lỗi tại đây: <https://eslint.org/docs/latest/rules/>

Tôi sẽ nêu ra một vài lỗi hay thấy:



| **Tên lỗi** | **Mô tả** | **Ví dụ** | **Sửa lỗi** |
| --- | --- | --- | --- |
| no-undef | Sử dụng biến chưa được khai báo. | console.log(x); | let x = 10;  console.log(x); |
| no-unused-vars | Biến được khai báo nhưng không sử dụng. | let y = 5; | let y = 5;  console.log(y); |
| eqeqeq | Dùng == thay vì === có thể gây lỗi logic. | if (a == '10') {} | if (a === '10') {} |
| semi | Thiếu dấu chấm phẩy ở cuối câu lệnh. | const name = 'Konan' | const name = 'Konan'; |
| no-console | Dùng console.log() trong mã nguồn. | console.log('Debugging'); | Xóa hoặc dùng logger như winston. |
| no-eval | Dùng eval() có thể gây lỗ hổng bảo mật. | eval("let z = 1;"); | Tránh dùng eval() và tìm cách thay thế. |
| no-loop-func | Định nghĩa hàm bên trong vòng lặp. | for (let i = 0; i < 5; i++) {  function foo() {}  foo();  } | Định nghĩa hàm bên ngoài vòng lặp. |
| no-var | Dùng var thay vì let hoặc const. | var count = 10; | let count = 10; |
| no-debugger | Từ khóa debugger còn sót lại trong mã. | debugger; | Xóa debugger. |
| no-new-wrappers | Dùng new String(), new Number(), v.v. không cần thiết. | let str = new String('Hello'); | let str = 'Hello'; |

Phần này cũng mang ý nghĩa giới thiệu, sau này nếu có áp dụng thì gặp lỗi nào tra lỗi đó sau.

# 6, Demo cách dùng ESLint

Theo như phần 1 thì có vài cách sử dụng ESLint. Vì tôi chỉ đang “cưỡi ngựa xem hoa” qua phần này nên tôi chỉ tìm hiểu demo cách hoạt động bằng cách sử dụng câu lệnh CLI và với những cấu hình mặc định (eslint:recommended, eslint:all,…)

Các bước thực hiện:

## 1, Cài đặt module ESLint vào dự án (ở đây dùng npm đi)

|  |
| --- |
| npm init @eslint/config@latest |

Chọn mấy cái phù hợp với dự án của mình:



Cài đặt xong thì ESLint sẽ tự tạo ra file eslint.config.mjs như sau:



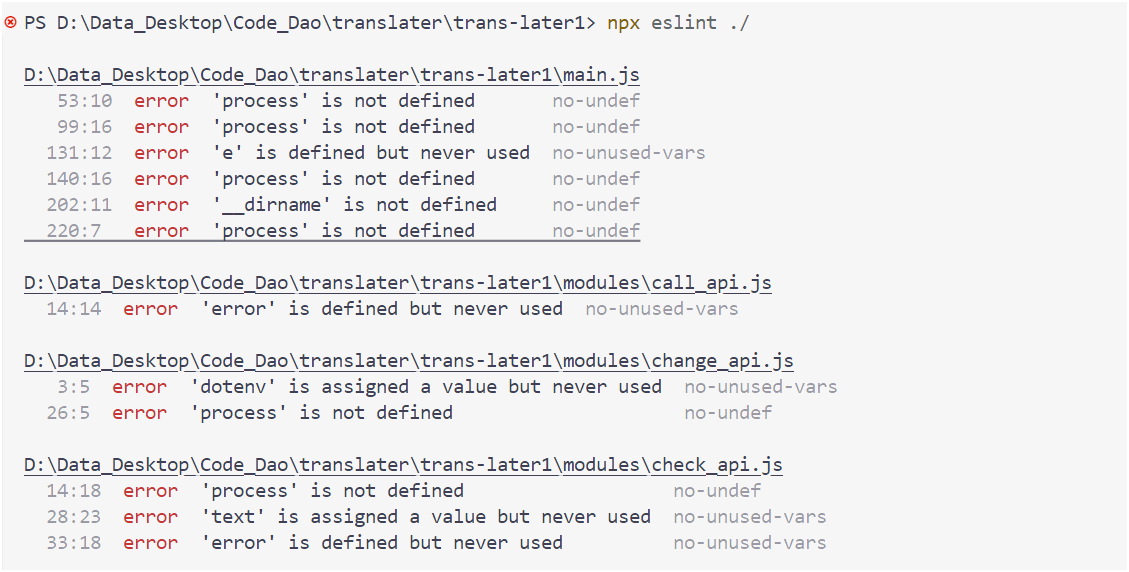
Như thế là cài đặt xong (cấu hình mặc định)

## 2, Tìm lỗi trong src code

|  |
| --- |
| npx eslint <đường\_dẫn\_đến\_file\_hoặc\_folder\_code> |

ESLint sẽ trả về lỗi cụ thể cho từng file.

Ví dụ: ở đây tôi để đường dẫn là “./”, nghĩa là phạm vi toàn ứng dụng.



## 3, Sửa lỗi tự động

Câu lệnh được thêm flag “--fix”:

|  |
| --- |
| npx eslint --fix <đường\_dẫn\_đến\_file\_hoặc\_folder\_code> |

Tuy nhiên câu lệnh này chỉ có thể tự sửa chữa những lỗi đơn giản, như lỗi style code. Còn lỗi hơi nặng một chút thì không thể tự sửa (và cũng không cần sửa làm gì, coder tự quyết định sửa như nào thì hơn)

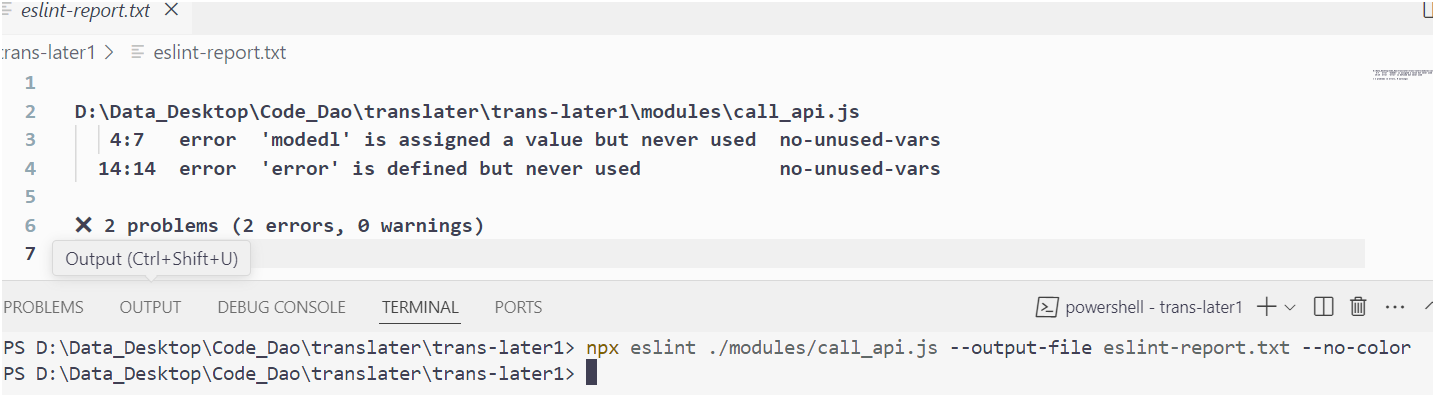
Danh sách các lỗi có thể sửa như sau: <https://eslint.org/docs/latest/rules/> (Các lỗi có thể sửa thì biểu tượng “🔧” sẽ sáng)

## 4, Xuất lỗi ra file

|  |
| --- |
| npx eslint <đường\_dẫn\_src\_code> --output-file <đường\_dẫn\_file\_log\_lỗi> --no-color |

Chú ý phải có flag “--no-color” vì terminal cố hiển thị mã màu ANSI khi xuất ra terminal nhưng khi xuất nó ra file thì nó hiển thị nguyên cái code mã màu.

VD:



Với câu lệnh trên thì ESLint output ra terminal như thế nào thì ghi vào file y hệt. Trừ màu.

Hết

*Do yêu cầu trong ngắn ngày và nhu cầu không nhiều nên tôi viết ngắn gọn để hiểu qua đã. Nếu sau này có nhu cầu học thêm thì sẽ viết thêm.*

*Tham khảo tutorial:* [*https://eslint.org/docs/latest/use/getting-started*](https://eslint.org/docs/latest/use/getting-started)

*Cảm ơn mọi người đã đọc tới đây!*