Engineer

ESLint

ein regelbasierter Linter für JavaScript

Was es ist

ESLint ist ein quelloffenes Werkzeug zur statischen Analyse. Es liest JavaScript-Quellcode, prüft ihn gegen einen konfigurierten Regelsatz und meldet jede Stelle, die eine Regel verletzt. Auf Wunsch repariert es einfachere Fälle selbst. Auf der Kommandozeile heißt der Befehl `eslint`; er wird über npm als Entwicklungsabhängigkeit des geprüften Projekts installiert. Nicholas C. Zakas begann das Projekt im Juni 2013. Es stellt sich weiterhin als konfigurierbarer JavaScript-Linter vor.

Das geprüfte Programm wird dabei nie ausgeführt. Ein Parser verwandelt jede Datei in einen abstrakten Syntaxbaum im ESTree-Format. Jede Regel ist ein kleines Modul, das für bestimmte Knotentypen aufgerufen werden möchte: jede Funktion, jeden Vergleich, jedes `await`. Die Regel untersucht den übergebenen Knoten und meldet bei einem Problem eine Nachricht samt Position. Manche Regeln enthalten auch eine Korrektur. Das Projekt akzeptiert solche Änderungen nur, wenn sie das Problem beheben, ohne das Verhalten des Programms zu ändern. `eslint --fix` wendet sie auf die Dateien an. Andere Regeln bieten stattdessen Vorschläge, die das Verhalten ändern können und deshalb einzeln von einem Menschen im Editor angewendet werden.

Fast alles um diesen Kern lässt sich austauschen. Plugins sind npm-Pakete, die Regeln, gemeinsam nutzbare Konfigurationen, Prozessoren oder Unterstützung für andere Sprachen ergänzen. Mit einem anderen Parser kann ESLint Syntax lesen, die der Standardparser nicht versteht, etwa TypeScript. Prozessoren extrahieren JavaScript aus anderen Dateien, beispielsweise aus Codeblöcken in einem Markdown-Dokument. Formatter ändern die Ausgabe des Berichts.
Zur eigenen Philosophie erklärt das Projekt, dass jede Regel eigenständig ist, sich abschalten lässt und unabhängig auf Warnung oder Fehler eingestellt werden kann.

All das wird in einer JavaScript-Datei im Projektstammverzeichnis konfiguriert, üblicherweise `eslint.config.js` oder einer Variante mit `.mjs` oder `.cjs`. Sie exportiert ein Array von Objekten. Jedes Objekt nennt die betroffenen Dateien sowie deren Regeln, globale Variablen und Parseroptionen; spätere Objekte überschreiben frühere. Dieser Aufbau heißt Flat Config und wurde mit ESLint v9.0.0 am 5. April 2024 zum Standard. ESLint v10.0.0, veröffentlicht am 6. Februar 2026, entfernte das ältere Format `.eslintrc` und die Datei `.eslintignore` vollständig.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ESLint?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ein Team führt eine Idee von der Skizze bis zu einem Produkt, bei dem Ihre Kunden sich anmelden können, statt sie zwischen drei Dienstleistern weiterzureichen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird.
GIS-Kartenprodukt zu 10-fachem Umsatz geführt — 10x Umsatzwachstum durch die Leitung des Kartenprodukts

GIS-Kartenprodukt zu 10-fachem Umsatz geführt Kontakt aufnehmen →

Warum es wichtig ist

Ein Linter rechtfertigt seinen Platz in dem Moment, in dem er das Zusammenführen einer Änderung verhindert. ESLint entscheidet darüber mit seinem Exitcode, und dieser richtet sich nach dem Schweregrad. Jede Regel steht auf `"off"`, `"warn"` oder `"error"`. Ein Fehler beendet den Befehl mit 1. Eine Warnung wird ausgegeben, der Befehl endet jedoch weiterhin mit 0, sofern `--max-warnings` keine Obergrenze setzt. Diese Obergrenze ist standardmäßig deaktiviert. Exitcode 2 ist einer fehlerhaften Konfiguration oder einem internen Absturz vorbehalten.

Die eigentliche Teamentscheidung bei ESLint lautet also, was eine Warnung bedeutet. Eine Warnung, die nie einen Build scheitern lässt, ist ein Hinweis. Stehen vierzig Zeilen davon im Terminal, scrollen Menschen darüber hinweg. Teams, deren Linter eine Grenze durchsetzen soll, setzen entweder jede wichtige Regel auf `"error"` oder führen die Qualitätsprüfung mit `--max-warnings 0` aus. In beiden Fällen besteht ein gesunder Branch die Lint-Prüfung ohne Meldungen. Eine einzelne neue Zeile fällt dann auf und wird gelesen.

Der Nutzen liegt darin, was JavaScript durchlässt. Die Sprache lädt eine Datei, die eine nicht deklarierte Variable verwendet, denselben Schlüssel zweimal in einem Objektliteral definiert, einen Wert mit `NaN` über `===` vergleicht oder hinter einem `return` Code enthält, der nie ausgeführt werden kann. Diese Probleme können das Laden überstehen und unbemerkt bleiben oder bei der Ausführung des Codes falsches Verhalten verursachen, auch in der Produktion. Die Regeln des empfohlenen ESLint-Regelsatzes erkennen alle vier vor dem Review. So kann die prüfende Person beim Lesen auf den Entwurf achten.

Es entstehen auch Kosten. In einer Konfiguration sammeln sich Meinungen. Eine Regel, die bei korrektem Code anschlägt, gewöhnt Menschen daran, die Ausgabe nicht mehr zu lesen; das ist schlechter als gar keine Regel. Ein Linter ist außerdem nur so gut wie der Zeitpunkt seiner Ausführung: Läuft er nur im Editor auf einigen Rechnern, prüft er nur einen Teil des Codes zu manchen Zeitpunkten.

Wie es in der Praxis aussieht

ESLint war einer von sechs Lintern hinter einer Null-Warnungen-Vorgabe für sechs Sprachen. Er prüfte TypeScript neben golangci-lint für Go, SQLFluff für SQL, Ruff für Python, ShellCheck für Shellskripte und markdownlint für Markdown. Durchgesetzt wurde die Vorgabe durch die oben beschriebene Schweregradentscheidung, einmal für alle: Jede Lintermeldung ist ein Fehler, es gibt keine Stufe „warn“ zum Ausweichen, und Unterdrückungen direkt im Quelltext sind verboten. Pre-Commit- und Pre-Push-Hooks führen Linter und Tests aus; Fehler innerhalb des abgedeckten Regel- und Testumfangs verhindern den jeweiligen Commit oder Push.

Dasselbe Repository brauchte anschließend Regeln, die kein veröffentlichtes Paket liefert: Regeln dafür, wie genau diese Codebasis funktionieren soll. Fünfzehn eigene ESLint-Regeln sichern die projektspezifischen Muster in TypeScript, neben 274 Prüfskripten, von denen 268 in die Commit-Hooks eingebunden sind. Einer dieser Validatoren prüft, ob jemand unbemerkt eine Unterdrückung im Quelltext ergänzt hat. Die Schutzprüfungen haben 146 eigene Tests, weil eine defekte Schutzprüfung alles durchlässt, ohne dass es jemand bemerkt. Die Seite benennt den Preis ausdrücklich: Commits dauern lange.

Diese Website legt für ihre eigenen Build-Skripte denselben Maßstab in kleinerem Umfang an. Ihre `eslint.config.mjs` beginnt mit `js.configs.recommended` und ergänzt nur Regeln, die der Code zum Messzeitpunkt bereits erfüllte. Kommentare dokumentieren eine erneute Messung am 2026-10-15 über alle 199

nicht als veraltet markierten Regeln, nach der 21 weitere Regeln für Korrektheit statt Geschmacksfragen aktiviert wurden. Jede Regel meldet Verstöße als Fehler. Auch `reportUnusedDisableDirectives`, standardmäßig eine Warnung, wird zum Fehler hochgestuft. Ein Disable-Kommentar, der nichts mehr unterdrückt, lässt den Lauf somit scheitern. Die Skripte deklarieren globale Variablen für Node und Browser gemeinsam, weil die an `page.evaluate` übergebenen Funktionsrümpfe im Browser laufen, wenige Zeilen neben Code, der in Node ausgeführt wird.

Wo es schiefgeht

Bei den meisten ESLint-Problemen tut der Linter genau das, was seine Konfiguration vorgibt. Das ist nicht immer das, was das Team darin zu lesen glaubt.

Der Build ist grün, und das Terminal ist voller Warnungen. Warnungen lassen den Exitcode bei 0. Die CI-Prüfung bleibt deshalb grün, während die Liste jede Woche um einige Zeilen wächst. Irgendwann steht darin ein echter Defekt, den niemand liest. Entscheiden Sie, welche Regeln wichtig sind, setzen Sie diese auf `"error"`, schalten Sie den Rest ab und ergänzen Sie `--max-warnings 0` in der Qualitätsprüfung, damit sich keine Warnungen mehr ansammeln können.

Ein Disable-Kommentar überlebt seinen Grund. Eine Zeile wurde einmal aus gutem Grund stummgeschaltet. Der Code darunter änderte sich, der Kommentar blieb und verdeckt nun das nächste Problem dieser Zeile oder gar nichts. Setzen Sie `linterOptions.reportUnusedDisableDirectives` auf `"error"`, damit Direktiven, die nichts unterdrücken, den Lauf scheitern lassen. Verlangen Sie bei den verbleibenden Kommentaren nach `--` eine Begründung, damit sie im Review beurteilt werden können:

```
// eslint-disable-next-line no-console -- printing is this CLI's output
console.log(report);
```

Ein Upgrade ändert die geprüften Dateien. Nach dem Wechsel zu v10 werden Pfade plötzlich gemeldet, die `.eslintignore` vorher übersprang, oder ein Verzeichnis folgt anderen Regeln. Version 10 berücksichtigt weder `.eslintrc.*` noch `.eslintignore` und beginnt die Suche nach `eslint.config.*` jeweils im Verzeichnis der geprüften Datei. Eine versehentliche Konfiguration in einem Unterverzeichnis bestimmt deshalb nun dessen Regeln. Übertragen Sie Ausnahmen in einen globalen Eintrag `ignores`, konvertieren Sie die alte Datei

mit `npx @eslint/migrate-config .eslintrc.json` und prüfen Sie das Ergebnis. Listen Sie anschließend jede `eslint.config.*` im Verzeichnisbaum auf und entfernen Sie die unbeabsichtigt verbliebenen Dateien.

Browsercode wird als undefiniert gemeldet. Ein Node-Skript, das Funktionen an einen Browser übergibt, erhält `no-undef`-Fehler für `document` und `window`. Jemand schaltet `no-undef` aus und verdeckt damit auch echte Tippfehler. Deklarieren Sie über `languageOptions.globals` für die jeweiligen Dateien jede Umgebung, in der der Code tatsächlich läuft.

Eine eigene Regel lässt alles durch. Nennt der Visitor einer Regel einen Knotentyp, der nie vorkommt, oder ist ihre Bedingung umgekehrt, meldet sie einfach nichts. Schweigen sieht genauso aus wie Regelkonformität. Testen Sie jede eigene Regel mit ESLints `RuleTester`, mit Code, der bestehen muss, und Code, der scheitern muss. Eine Regel, die keine Meldungen mehr liefert, scheitert dann am eigenen Test, bevor sie der Codebasis schadet.

Weiterführende Lektüre

- Grundbegriffe — Regeln, Korrekturen, Plugins, Parser und Prozessoren nach den Definitionen des Projekts.
- Konfigurationsdateien — das Flat-Config-Array, Dateimuster, globale Ausnahmen und das Zusammenführen von Objekten.
- Regeln konfigurieren — Schweregrade, Regeloptionen und Kommentare, die eine Regel für eine Zeile deaktivieren.
- Referenz der Kommandozeilenschnittstelle — sämtliche Optionen, darunter `--max-warnings`, und die Bedeutung jedes Exitcodes.
- ESLint v10.0.0 veröffentlicht — die Entfernung von eslintrc und die neue Konfigurationssuche in den Versionshinweisen.
- Eigene Regeln — eine Regel für den Syntaxbaum schreiben und mit `RuleTester` nachweisen, dass sie funktioniert.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ESLint?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/eslint/