

# JavaScript



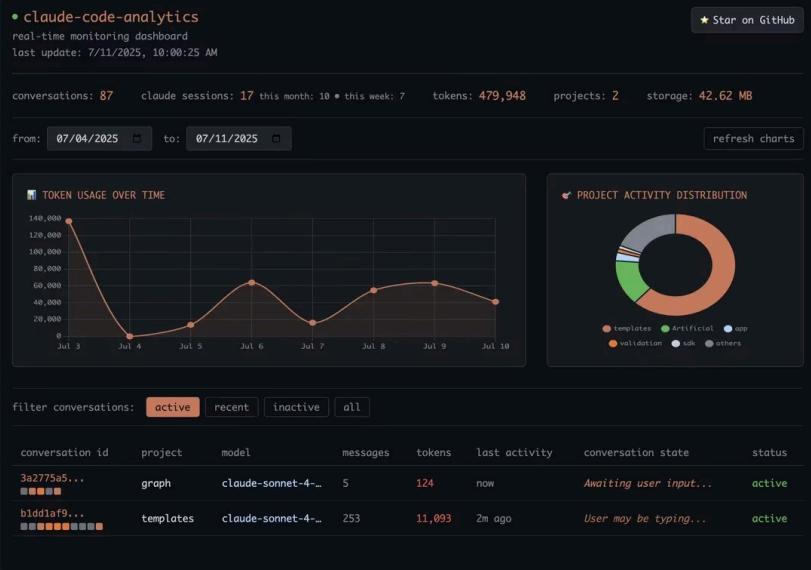
## ESLint

Статичний аналіз коду для JavaScript

# Що таке ESLint?

ESLint — це потужний інструмент статичного аналізу коду для JavaScript, який допомагає виявляти потенційні проблеми в коді та рекомендує стандартизовані підходи до написання JavaScript.

Цей інструмент аналізує ваш код ще до його виконання, знаходячи помилки, небезпечні конструкції та порушення стилю кодування. ESLint став стандартом індустрії для забезпечення якості JavaScript-коду.



# Основні особливості ESLint



## Виявлення помилок

Аналізує код на предмет синтаксичних помилок, потенційних проблем та стилістичних недоліків



## Гнучка конфігурація

Налаштування правил для відповідності специфічним стандартам вашого проекту чи команди



## Розширюваність

Широкий набір плагінів та можливість створення власних правил

## Інтеграція з редакторами

Підтримка більшості IDE дозволяє використовувати ESLint безпосередньо в процесі написання коду

## Популярні стандарти

Підтримка відомих стандартів: Airbnb, Standard, Google та інші

ESLint є потужним інструментом для покращення якості JavaScript-коду, допомагає виявляти й усувати помилки та забезпечує дотримання консистентного стилю коду у всьому проекті.

```
Collinsworth@Joshs-MacBook-Pro:~/Projects/joco-grid$ npm i
t:(master) npm i
, and audited 2456 packages in 16s
ooking for funding
or details
(46 moderate, 42 high, 5 critical)
hat do not require attention, run:
es possible (including breaking ch
orce
view, and may require choosing
ency.
details.
t:(master) x
```

# Інсталяція ESLint

Встановлення ESLint локально у вашому проекті відбувається за допомогою простої команди:

```
npm init @eslint/config@latest
```

01

## Запуск інтерактивної CLI

Команда запускає програму налаштування з питаннями про ваш проект

02

## Вибір мови програмування

Вкажіть, чи використовуєте ви JavaScript, TypeScript або інші варіанти

03

## Середовище виконання

Оберіть браузер, Node.js або обидва варіанти

04

## Набір правил

Виберіть з готових конфігурацій або створіть власну

# Конфігурація ESLint

Після встановлення створюється конфігураційний файл **eslint.config.js** або **.eslintrc.js**, який містить налаштування для вашого проекту.

Приклад базової конфігурації:

- Саме тут визначаються правила, які будуть застосовані до вашого коду

```
import js from "@eslint/js";

export default [
  js.configs.recommended,
  {
    rules: {
      "no-unused-vars": "warn",
      "no-undef": "warn"
    }
  }
];
```

[Опції конфігурації](#)[Доступні правила](#)

# Суворість правил

ESLint дозволяє налаштовувати рівень серйозності для кожного правила, що забезпечує гнучкість у процесі розробки.

## "off" або 0

Вимкнути правило повністю

## "warn" або 1

Попередження без впливу на код виходу

## "error" або 2

Помилка з кодом виходу 1

## Простий приклад

```
{  
  "rules": {  
    "semi": "error",  
    "no-console": "warn",  
    "no-unused-vars": "off"  
  }  
}
```

## Розширений приклад

```
{  
  "rules": {  
    "indent": ["error", 2, {  
      "SwitchCase": 1  
    }],  
    "quotes": ["warn", "double", {  
      "avoidEscape": true  
    }]  
  }  
}
```

Правила зазвичай мають значення **"error"** для забезпечення відповідності під час CI/CD, перевірок перед комітом та пул-реквестів.

# Плагіни та ігнорування файлів

## Підключення плагінів

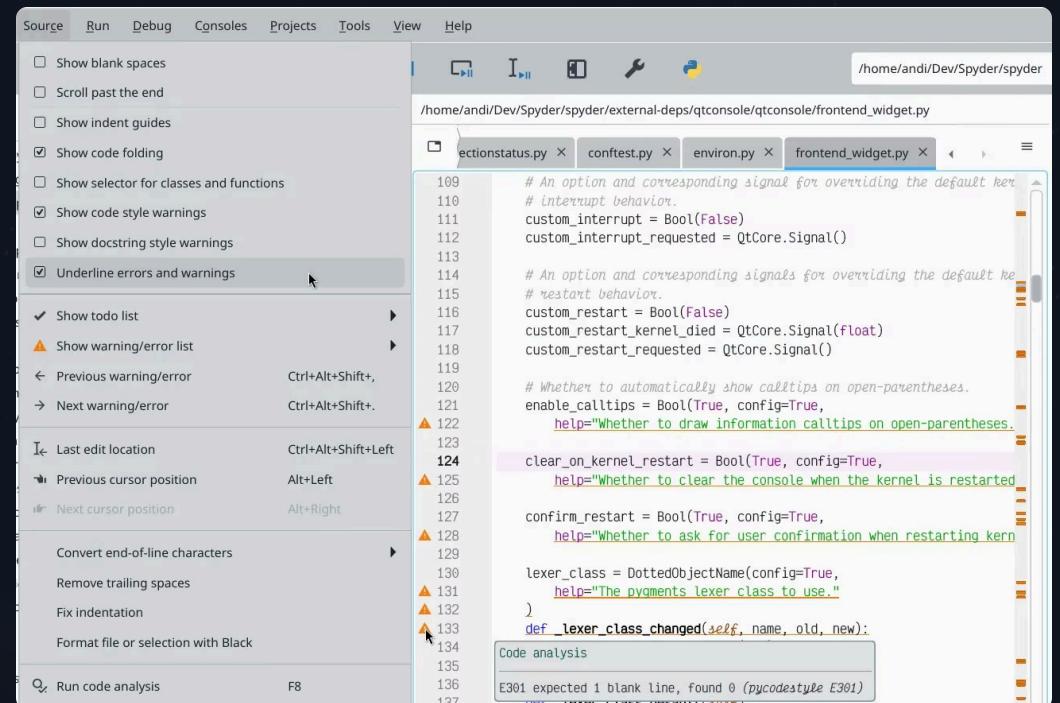
ESLint підтримує сторонні плагіни для розширення функціональності. Встановіть їх через npm та налаштуйте у конфігурації:

```
{  
  "plugins": [  
    "plugin1",  
    "eslint-plugin-plugin2"  
  ]  
}
```

## Файл .eslintignore

Створіть файл **.eslintignore** для виключення файлів та папок з перевірки:

```
# Ignore artifacts:  
build  
coverage  
  
# Dependencies  
node_modules/
```



ESLint автоматично знаходить та підсвічує помилки у вашому коді, допомагаючи підтримувати високу якість розробки.

# Запуск з командного рядка

## Автоматичне виправлення

ESLint може автоматично виправляти деякі помилки у вашому коді

Для запуску автоматичного виправлення виконайте команду:

```
npx eslint --fix .
```

Ця команда перевірить усі файли в поточній директорії та автоматично виправить помилки, які можна виправити програмно.

## Більше можливостей CLI

ESLint пропонує широкий набір опцій командного рядка для різних сценаріїв використання. Перегляньте повний список опцій в офіційній документації.

[Опції командного рядка](#)

# Інтеграція з IDE

## WebStorm

Вбудована підтримка ESLint. Перейдіть у налаштування та увімкніть автоматичну перевірку після встановлення ESLint та налаштування конфігурації.

Інтеграція з IDE забезпечує миттєвий зворотний зв'язок під час написання коду, підсвічуєчи помилки та попередження безпосередньо в редакторі.

## Visual Studio Code

Встановіть плагін "**Prettier - Code formatter**" з магазину розширень для інтеграції з ESLint.

[Детальна інформація](#)

# Команди для запуску лінтера

-debug-from-run",

"/www",  
ode --inspect-brk ./bin/www  
ntrc.json"

1.4.4",

',  
6.3",

"

## Базова перевірка

1

```
npx eslint .
```

Перевіряє всі файли в проекті

## Перевірка конкретного файлу

2

```
npx eslint src/index.js
```

Аналізує окремий файл

## Автоматичне виправлення

3

```
npx eslint --fix .
```

Виправляє помилки автоматично

## Вивід у форматі JSON

4

```
npx eslint . --format json
```

Корисно для CI/CD інтеграції

Додайте ці команди в **package.json** scripts для зручного використання в команді.

# Приклад: Cypress з TypeScript

Конфігурація ESLint для проекту автоматизації тестування на Cypress з використанням TypeScript:

```
module.exports = {  
  extends: [  
    'eslint:recommended',  
    'plugin:@typescript-eslint/recommended',  
    'plugin:cypress/recommended'  
  ],  
  parser: '@typescript-eslint/parser',  
  plugins: ['@typescript-eslint', 'cypress'],  
  env: {  
    'cypress/globals': true  
  },  
  rules: {  
    '@typescript-eslint/no-unused-vars': 'warn',  
    'cypress/no-unnecessary-waiting': 'error',  
    'cypress/assertion-before-screenshot': 'warn'  
  }  
}
```



## Підтримка TypeScript

Використання парсера та плагіна для TypeScript



## Cypress-специфічні правила

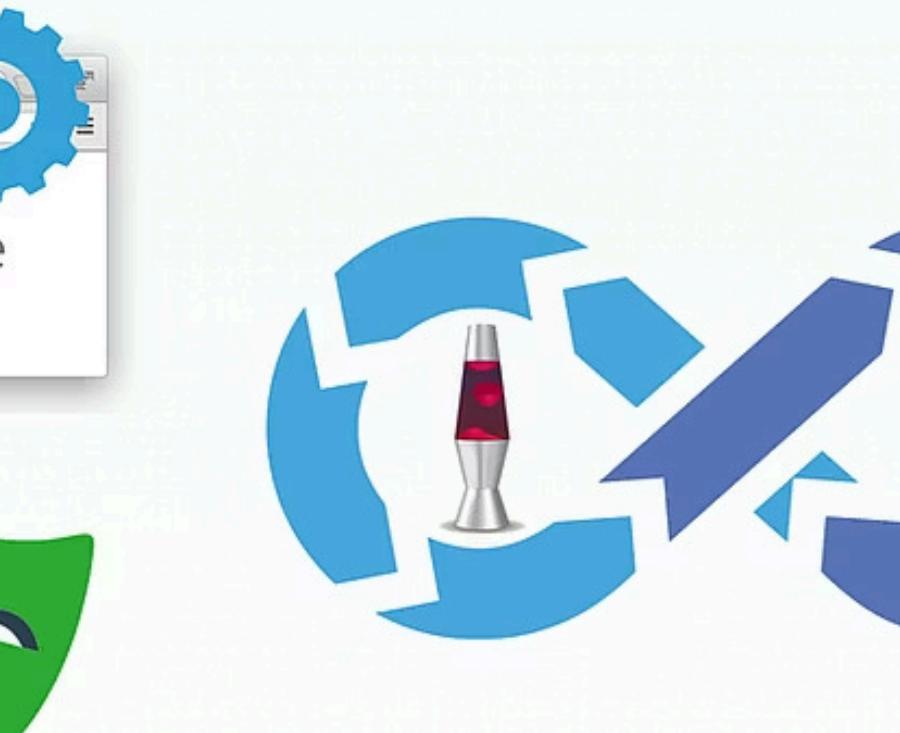
Плагін cypress додає правила для тестів



## Налаштування середовища

Глобальні змінні Cypress доступні без помилок

# *s Testing with*



## *Part 1: Setup*

## Приклад: Playwright з TypeScript

Конфігурація ESLint для проекту на Playwright з TypeScript забезпечує якість тестового коду:

```
module.exports = {  
  extends: [  
    'eslint:recommended',  
    'plugin:@typescript-eslint/recommended',  
    'plugin:playwright/recommended'  
  ],  
  parser: '@typescript-eslint/parser',  
  parserOptions: {  
    ecmaVersion: 2021,  
    sourceType: 'module'  
  },  
  plugins: ['@typescript-eslint', 'playwright'],  
  rules: {  
    'playwright/no-wait-for-timeout': 'error',  
    'playwright/missing-playwright-await': 'error',  
    '@typescript-eslint/explicit-function-return-type': 'off',  
    'playwright/no-element-handle': 'warn'  
  }  
}
```

- ❑ Playwright plugін надає специфічні правила для виявлення типових помилок у тестах, таких як відсутність await або використання застарілих методів.

# Before

```
<div className="frame1">
  <div className="card">
    <img className="avatar" />
    <div className="title">W...
  </div>
  <div className="group8">
    <div className="descript...
    <button className="butto...
  </div>
</div>
```

# After

```
<Card>
  <CardHeader>
    <Avatar src={userImage} />
    <Typography variant="h5" />
  </CardHeader>
  <CardContent>
    <Typography variant="body...
    <Button variant="contai...
  </CardContent>
</Card>
```

## Порівняння Prettier & ESLint

Prettier та ESLint — це два різні інструменти, які мають різні функції, але часто використовуються разом для поліпшення якості коду.

### Prettier

- **Автоформатування**

Автоматичне форматування коду для однорідності стилю

- **Багатомовність**

Підтримка JavaScript, HTML, CSS, JSON, YAML та інших

- **Форматування файлу**

Обробляє весь файл без можливості вибору окремих правил

### ESLint

- **Статичний аналіз**

Перевірка синтаксису, виявлення помилок та стилістичних проблем

- **Гнучкість**

Налаштування різних правил для різних типів проектів

- **Розширення**

Плагіни та кастомні правила для специфічних потреб

# Використання Prettier та ESLint разом



## Prettier

Форматування коду для однорідності та читабельності



## ESLint

Статичний аналіз та виявлення проблем згідно з правилами



## Якість коду

Високий стандарт у проектах

Обидва інструменти можуть використовуватися разом для забезпечення якості коду. Prettier відповідає за візуальне форматування, а ESLint — за логічну коректність та дотримання стандартів.



### ⚠️ Важливо

ESLint має велику кількість правил, і деякі з них можуть конфліктувати з правилами Prettier. Використовуйте **eslint-config-prettier** для вимкнення конфліктуючих правил ESLint.

