

LAGASH.



UNIVERSITY

ESlint Prettier



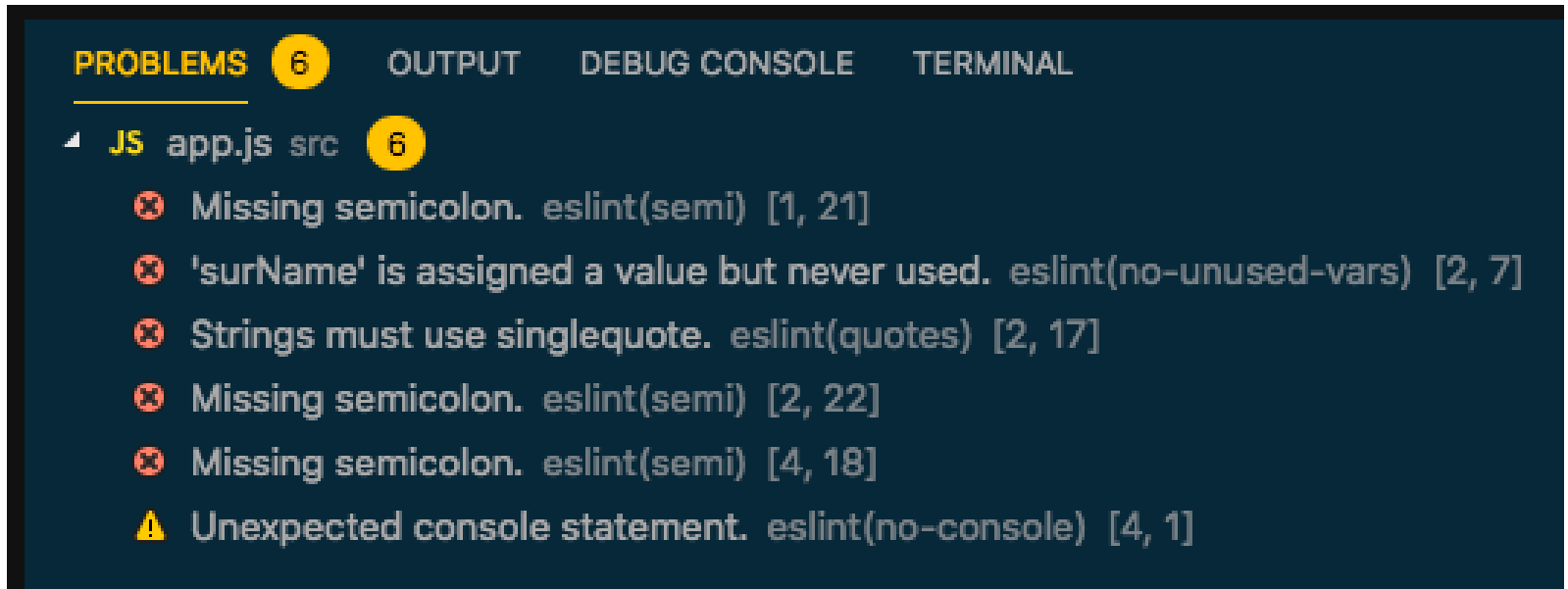
Conceptos Previos

- **Linters:** Herramientas automáticas que permiten aumentar la calidad del código.
- **Eslint:** Linter que identifica patrones problemáticos en el código.(Problemas de compilación o futuros bugs)
- **Prettier:** Formateador de código. Permite que el código se ajuste a un determinado estilo.

¿Cómo trabaja ESLint?

- Cuando encuentra un problema lo arregla automáticamente o advierte de ello en mensajes.

- Para realizar el proceso se basa en una serie de reglas previamente configuradas y cargadas.



The screenshot shows the 'PROBLEMS' panel in a code editor. It lists 6 errors found in 'app.js' at 'src'. The errors are:

- Missing semicolon. eslint(semi) [1, 21]
- 'surName' is assigned a value but never used. eslint(no-unused-vars) [2, 7]
- Strings must use singlequote. eslint(quotes) [2, 17]
- Missing semicolon. eslint(semi) [2, 22]
- Missing semicolon. eslint(semi) [4, 18]
- Unexpected console statement. eslint(no-console) [4, 1]

The panel has tabs for PROBLEMS (6), OUTPUT, DEBUG CONSOLE, and TERMINAL. The 'PROBLEMS' tab is selected, showing a list of errors with their respective line and column numbers.

¿Cómo trabaja Prettier?

```
foo(reallyLongArg(), omgSoManyParameters(), IShouldRefactorThis(), isThereSeriouslyAnotherOne())
```



Elimina el estilo del código original

```
foo(  
  reallyLongArg(),  
  omgSoManyParameters(),  
  IShouldRefactorThis(),  
  isThereSeriouslyAnotherOne()  
);
```



Y se asegura que el generado se ajuste a un estilo consistente

Instalación

```
npm install eslint --save-dev
```

```
# or
```

```
yarn add eslint --dev
```

Luego debe configurar un archivo de configuración:

```
$ npx eslint --init
```

Después de eso, puede ejecutar ESLint en cualquier archivo o directorio como este:

```
$ npx eslint yourfile.js
```

Configuración

Después de ejecutar `eslint --init`, tendrá un `.eslintrc` archivo en su directorio. En él, verá algunas reglas configuradas de esta manera:

```
{  
  "rules": {  
    "semi": ["error", "always"],  
    "quotes": ["error", "double"]  
  }  
}
```

Los nombres "semi" y "quotes" son nombres de las reglas en ESLint. El primer valor es el nivel de error de la regla y puede ser uno de estos valores :

- "off" o 0 - apague la regla
- "warn" o 1 - active la regla como advertencia (no afecta el código de salida)
- "error" o 2 - active la regla como un error (el código de salida será 1)



Además de reglas, Eslint puede configurar variables globales y entornos

Ventajas de Trabajar con este Método

Código Limpio y
claro de entender

Detecta errores de
código o sintaxis
más fácilmente.

Codea bajo buenas
prácticas.

Mantén un estilo
consistente de
código junto a tu
equipo.

Evita comitear
código no valido
(ej. console.log)

Salva tu tiempo y
energía al
momento de
refactorizar código.

Lint Staged

Usando `lint-staged` podremos estar seguros de que el código que subimos a Github o cualquier otro repositorio de Git siempre este revisado por ESLint y Prettier.

Instalación:

`yarn add lint-staged husky`

Configuración de Lint Staged:

```
{  
  "*.js": ["eslint --fix", "git add"]  
}
```

Nuevo Script en el `package.json`:

```
"scripts": { "precommit": "lint-staged", },
```

- Con esta nueva configuración en nuestro proyecto, podemos estar seguros de que el código que escribimos junto con nuestro equipo de desarrollo siempre estará linteado por ESLint y Prettier gracias a Lint Staged.