Proyecto final: Informe código limpio

Sebastian Andrade Cedano
Cristian Steven Motta Ojeda
Juan Esteban Santacruz Corredor

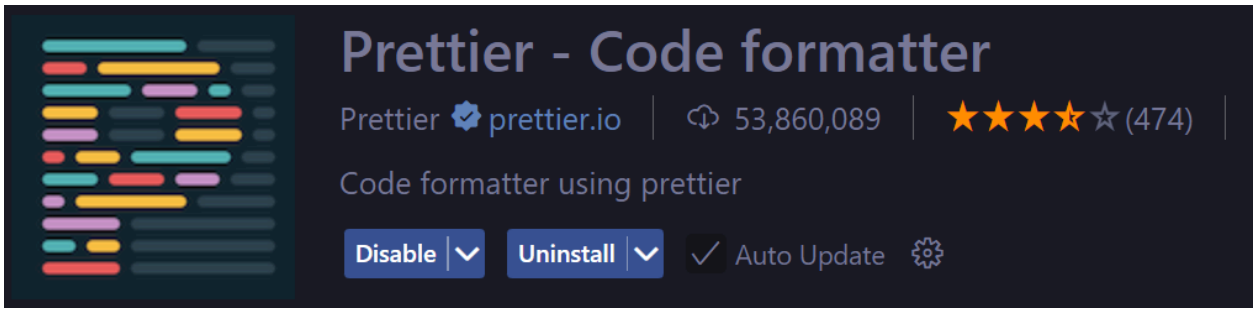
Ingeniería de software I

Universidad Nacional de Colombia

Bogotá D.C.
2025

Para configurar las herramientas necesarias que nos ayudarán a aplicar los principios de Clean Code en nuestro proyecto, comenzaremos revisando las extensiones que utilizamos en Visual Studio Code.

Prettier

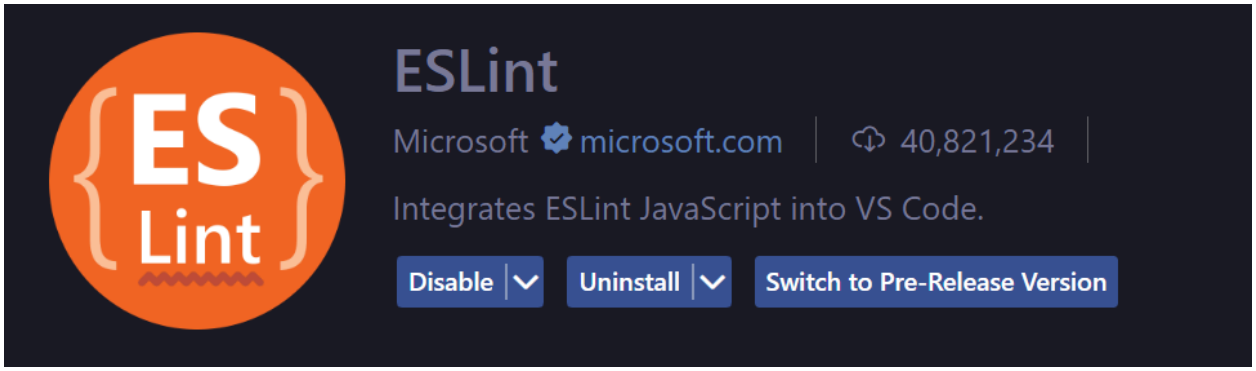


Prettier - Code formatter es una extensión para Visual Studio Code que formatea automáticamente el código en varios lenguajes como JavaScript, TypeScript, HTML, CSS, JSON y más.

Dentro de sus funciones principales se encuentran:

- Formato automático: Reestructura el código siguiendo reglas predefinidas para mejorar la legibilidad.
- Soporte para múltiples lenguajes: Compatible con HTML, CSS, JavaScript, TypeScript, JSON, YAML, Markdown, etc.
- Integración con VS Code: Permite formateo al guardar o mediante atajos de teclado.
- Configuración personalizable: Admite ajustes como uso de comillas simples/dobles, tamaño de tabulación y punto y coma opcional.
- Compatibilidad con linters: Funciona junto a ESLint (perfecto para nuestro proyecto por que es una herramienta que vamos a implementar) y otras herramientas de análisis de código.

ESLint



ESLint es una extensión para Visual Studio Code que ayuda a detectar y corregir errores en el código JavaScript y TypeScript siguiendo reglas de estilo y buenas prácticas.

Dentro de sus funciones principales se encuentran:

- **Análisis estático**: Identifica errores y problemas potenciales en el código.
- **Corrección automática**: Puede arreglar errores y problemas de estilo automáticamente.
- **Soporte para reglas personalizadas**: Permite configurar reglas específicas o usar estándares como Airbnb o Google.
- **Integración con Prettier**: Puede trabajar junto a Prettier para un formato y estilo de código consistentes.
- **Compatibilidad con múltiples entornos**: Funciona con proyectos en Node.js, React, Vue, entre otros.

A Continuación vamos a presentar el paso a paso para integrar ESLint a nuestro proyecto. Para esto vamos a hacer una breve explicación de cuál es la diferencia entre la extensión de la extensión en vsc y la instalación en nuestro proyecto y finalmente un paso a paso para dejarlo completamente funcional (todos estos pasos fueron los que seguimos para nuestro proyecto).

La extensión ESLint de Visual Studio Code y la instalación de ESLint en el proyecto son complementarias, pero tienen funciones diferentes. Por una parte ESLint en el proyecto (al instalarlo con npm i) tiene algunas características específicas:

- Se instala localmente en el proyecto.
- Permite ejecutar ESLint en la terminal (npx eslint .).
- Se configura com regras específicas en eslint.config.js
- Se asegura de que todos los desarrolladores del proyecto usen las mismas reglas.

Por otra parte la extensión de ESLint en VS Code

- Proporciona resaltado de errores en tiempo real dentro del editor.
- Usa la configuración de ESLint del proyecto si existe.
- Facilita la corrección automática en la edición (Format on Save).
- No funciona sin un ESLint instalado en el proyecto.

Con esto en cuenta veamos el paso a paso para la instalación de ESLint en **nuestro proyecto**

Para empezar es necesario instalar ESLint y las dependencias correspondientes con el siguiente comando en consola:

`npm install eslint @eslint/js globals eslint-plugin-react-hooks eslint-plugin-react-refresh typescript-eslint --save-dev`

Una breve descripción de lo que contiene el comando

- **eslint** → La herramienta principal para análisis de código.
- **@eslint/js** → Configuración recomendada para JavaScript en ESLint.
- **globals** → Define variables globales según el entorno (ej. window, document, process).
- **eslint-plugin-react-hooks** → Reglas para asegurar el uso correcto de los hooks de React.
- **eslint-plugin-react-refresh** → Reglas para optimizar el **Fast Refresh** en entornos de desarrollo de React.
- **typescript-eslint** → Conjunto de herramientas que permite que ESLint entienda TypeScript y aplique reglas específicas.

A Continuación se crea el archivo de configuración ***'eslint.config.js'*** en la raíz del proyecto. Una posible configuración que puede tener este archivo es el siguiente:

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import tseslint from 'typescript-eslint'

export default tseslint.config(
  { ignores: ['dist'] },
  {
    extends: [js.configs.recommended, ...tseslint.configs.recommended],
    files: ['**/*.{ts,tsx}'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
    rules: {
      ...reactHooks.configs.recommended.rules,
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
  },
)
```

Archivo eslint.config.js usado en el proyecto

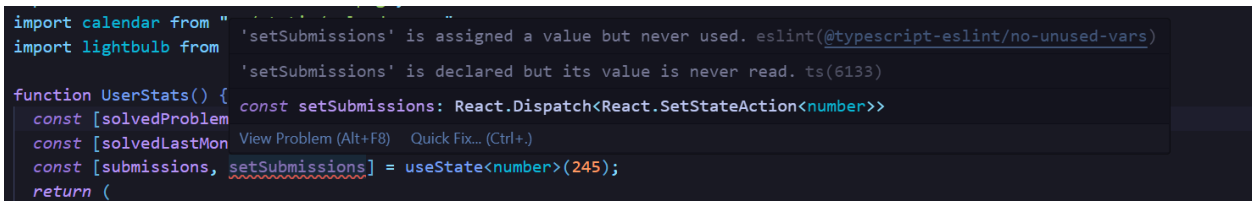
En este archivo se especifican las configuraciones para nuestro linters en términos generales se especifica lo siguiente:

- Habilita reglas para TypeScript y JavaScript.
- Aplica reglas recomendadas para React y Hooks.
- Ignora la carpeta dist para evitar revisar archivos compilados.
- Define un entorno de navegador.
- Activa reglas para evitar problemas con Fast Refresh en React.

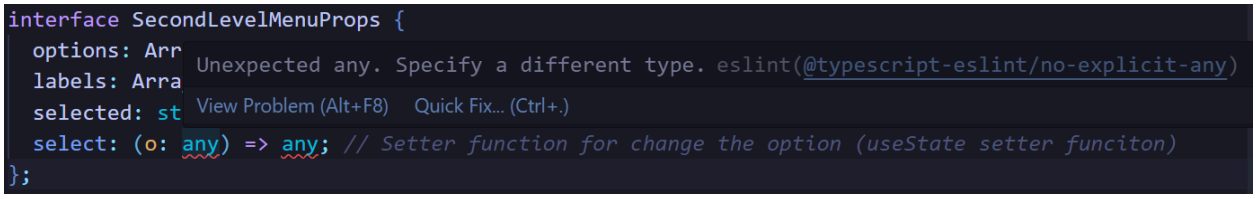
Finalmente se puede ejecutar ESLint de dos maneras, para compilar una vez y detecte los errores, o que corrija los errores automáticamente

- ***npx eslint .***
- ***npx eslint . --fix***

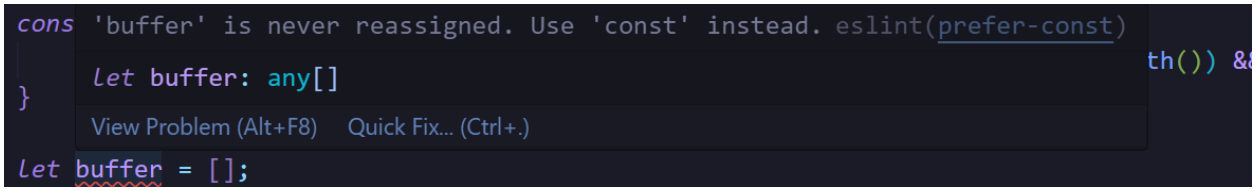
Finalmente se deberían mostrar diferentes errores de nuestro código que no afectan su ejecución pero si están relacionados con el clean code.



Error en un estado que nunca es utilizado (React)



Problema con el tipado de una interface (Ts)



Malas prácticas al declarar constantes como variables (Ts / Js)

Como se muestra en las imágenes, los errores no están relacionados con problemas de lógica o sintaxis, sino con malas prácticas detectadas por ESLint. Aunque no afectan la ejecución del código, pueden dificultar el desarrollo del proyecto.

NOTA: Como se está utilizando typeScript en todo el proyecto (backend y frontend) esta configuración sirve para ambos.

Narraciones sobre el código del equipo

1. *Sebastian Andrade Cedano:* Hubo algunos inconvenientes con la base de datos; sin embargo, no representaron un problema mayor, ya que, gracias a la comunicación constante del equipo, se solucionaron fácilmente. Actualmente, cada miembro cumple con sus tareas asignadas, pero todos estamos involucrados en el desarrollo del proyecto, por lo que no hay problema si se tuviera que modificar el código.

2. *Cristian Steven Motta Ojeda:* El código ha sido fácil de trabajar y leer. Al inicio, distribuimos las tareas según las fortalezas y debilidades de cada miembro. Sin embargo, para salir de nuestra zona de confort, asumimos algunas tareas fuera de nuestras especialidades. Además, mantenemos una comunicación constante y sabemos exactamente en qué está trabajando cada integrante del equipo.

3. *Juan Esteban Santacruz Corredor:* Aunque faltan comentarios, el código sigue buenas prácticas para el nombramiento de variables y modularización de la lógica por lo que es intuitivo. Todos nos distribuimos tareas al inicio y sabemos en qué parte del avance está cada uno. Hasta el momento aunque se han presentado dificultades hemos podido lograr los plazos acordados.