Testing - Linter

Esteban Barrera Sanabria

Maria Camila Castro Porras

Esteban Garcia Gaitan

Tomas Saldana Leguizamo

Facultad de Ingeniería

Universidad Nacional de Colombia

Ingeniería de Software I

Oscar Eduardo Alvarez Rodriguez

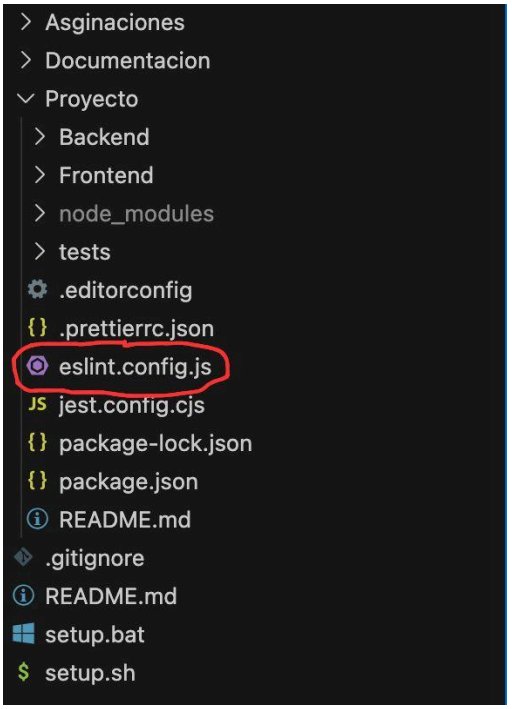
Noviembre 2025

1. Introducción

En el proyecto PocketVet se utilizó ESLint como analizador estático de código, muy frecuente usado para JavaScript/TypeScript

Esta herramienta permite revisar el código fuente sin ejecutarlo, con el fin de detectar errores comunes, inconsistencias de estilo y malas prácticas antes de que lleguen a producción. En otras palabras, su propósito es mantener un código más limpio, uniforme y fácil de mantener.

ESLint se ejecutó sobre la carpeta del proyecto y permitió identificar posibles problemas de React Native en Frontend principalmente.



El archivo contiene las configuraciones (reglas de código) descritas a continuación

```javascript
import eslintPluginPrettier from "eslint-plugin-prettier";
import eslintPluginReact from "eslint-plugin-react";
import eslintPluginReactHooks from "eslint-plugin-react-hooks";

export default [
  {
    files: ["**/*.{js,jsx,ts,tsx}"],
    ignores: ["node_modules/", "dist/", "build/", "expo-env.d.ts"],
```

```
    languageOptions: {
      parserOptions: {
        ecmaVersion: 2022,
        sourceType: "module",
      },
    },
    plugins: {
      react: eslintPluginReact,
      "react-hooks": eslintPluginReactHooks,
      prettier: eslintPluginPrettier,
    },
    rules: {
      "react/react-in-jsx-scope": "off",
      "react-hooks/rules-of-hooks": "error",
      "react-hooks/exhaustive-deps": "warn",
      "prettier/prettier": "warn",
      "no-unused-vars": "warn",
      "no-console": "off",
    },
    settings: {
      react: {
        version: "detect",
      },
    },
  },
];
```

En las configuraciones de arriba por ejemplo, se integra Prettier, también bastante usado en JavaScript para IDE. Además, entre otras reglas, impone estas:

- No exige importar React manualmente (react/react-in-jsx-scope: off): Desde que la nueva versión de React ya no requiere import React, el linter evita marcarlo como error.
- Uso correcto de hooks (react-hooks/rules-of-hooks: error): Garantiza que useState, useEffect y otros hooks solo se usen donde es permitido, básicamente no dentro de loops o condiciones.
- Dependencias completas en efectos (react-hooks/exhaustive-deps: warn): Advierte cuando un useEffect tiene dependencias incompletas para evitar comportamientos inesperados.
- Advertencia si hay variables sin usar (no-unused-vars: warn): Evita dejar código muerto o imports innecesarios.
- Permite usar console.log sin marcar error (no-console: off): Deja habilitados los logs para debug durante el desarrollo.

2. Configuración del Entorno y Herramientas Utilizadas

2.1 Herramientas de Análisis

- **ESLint (v9.39.1)**
  Utilizado para detectar errores de sintaxis, malas prácticas, problemas de React/React Hooks y uso incorrecto de variables.
- **Prettier**
  Aplicado para garantizar consistencia en el formato del código (indentación, comillas, espaciado, saltos de línea, etc.).

2.2 Problemas Iniciales Detectados

Durante la primera ejecución del linter, se identificó el siguiente error:

```
JavaScript

Error [ERR_MODULE_NOT_FOUND]: Cannot find package 'eslint-plugin-prettier'
```

Este problema impidió la ejecución completa del análisis debido a dependencias faltantes requeridas por la configuración del proyecto.

2.3 Corrección de Dependencias

Para resolver el error y permitir la ejecución del análisis, se instalaron las siguientes dependencias:

```
JavaScript

npm install --save-dev eslint-plugin-prettier eslint-plugin-react
eslint-plugin-react-hooks
```

3. Configuración Aplicada en el Análisis

3.1 Configuración de ESLint (eslint.config.js)

La configuración utilizada define reglas específicas para archivos JavaScript, TypeScript, JSX y TSX, incluyendo:

- Reglas de React y React Hooks
- Integración con Prettier para formato
- Restricciones sobre variables no usadas
- Detección de dependencias faltantes en useEffect
- Ignorar carpetas como node_modules, dist y build

3.2 Configuración de Prettier (.prettierrc.json)

Incluye reglas relacionadas con estilo y formato, tales como:

- Uso de comillas simples
- Semicolons habilitados
- printWidth de 100 caracteres
- Estilo consistente de arrow functions
- Forzar saltos de línea estilo UNIX

4. Ejecución del Análisis Estático del Proyecto PocketVet

Para evaluar la calidad, consistencia y estado actual del código del proyecto PocketVet, se ejecutó un análisis estático integral sobre todo el repositorio, incluyendo Backend, Frontend, configuración y tests.

Comandos Ejecutados

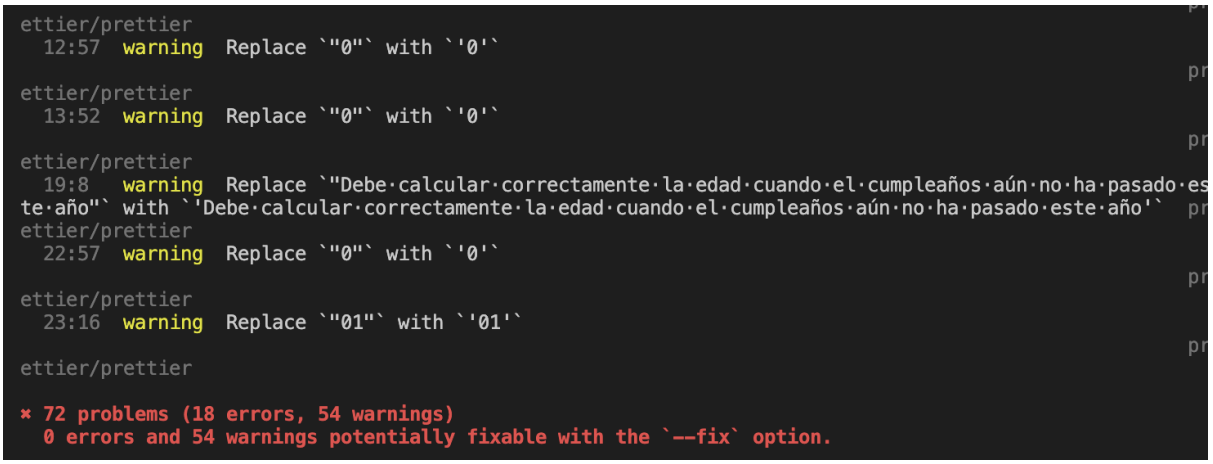
Se ejecutaron los siguientes comandos desde la carpeta raíz del proyecto:

```javascript
npx eslint . --ext .js,.jsx,.ts,.tsx
npx prettier --check .
```

La ejecución cubrió todos los archivos JavaScript, TypeScript y TSX del repositorio completo.

5. Resultados del Análisis Estático

Los resultados evidencian problemas tanto de formato como de parsing, afectando al Backend, Frontend y archivos de configuración.



```
[warn] Frontend/app/pet/upload.tsx
[warn] Frontend/index.tsx
[warn] Frontend/package.json
[warn] Frontend/src/declarations.d.ts
[warn] Frontend/src/services/api.ts
[warn] Frontend/tsconfig.json
[warn] jest.config.cjs
[warn] README.md
[warn] tests/calculatePetAge.test.ts
[warn] tsconfig.json
[warn] Code style issues found in 29 files. Run Prettier with --write to fix.
```

5.1 Estadísticas generales (ESLint)

Métrica	Resultado
Total de problemas	72
Errores	18
Advertencias	54
Archivos con problemas	27 archivos
Problemas potencialmente corregibles con --fix	54 (solo formato)

5.2 Tipo de errores encontrados

a) Parsing Errors (TypeScript y React)

18 errores graves por errores de parsing:

- Unexpected token :
- Unexpected token <
- The keyword 'interface' is reserved
- Unexpected token module

Archivos más afectados (Frontend):

- Frontend/app/auth/login.tsx
- Frontend/app/layout.tsx
- Frontend/index.tsx
- Frontend/app/main/home.tsx
- Frontend/app/pet/carnet.tsx
- Frontend/src/declarations.d.ts

b) Advertencias de ESLint

54 advertencias clasificadas principalmente en:

Categoría	Conteo
Comillas inconsistentes	+30
Espaciado y estructura	+15
Import/export inconsistente	+9
Console.logs y estilos	+4

Archivos con más advertencias (Backend)

- Backend/prisma.config.ts
- Backend/server.js
- Backend/src/controllers/*
- Backend/src/routes/*

6. Arreglar errores

Comandos Ejecutados

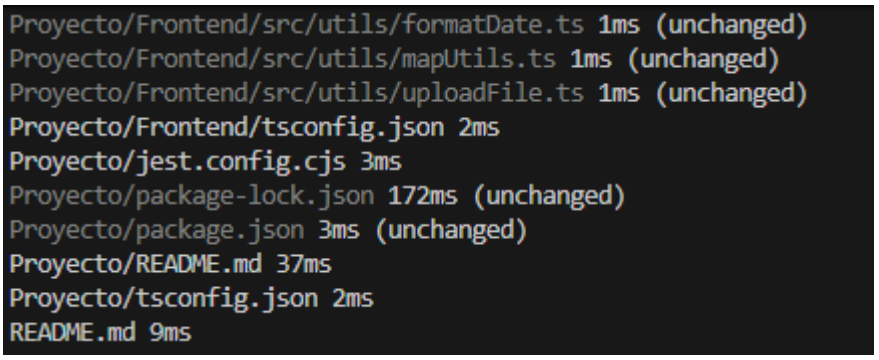
Para arreglar los códigos se ejecutó el comando

```
JavaScript
npx prettier . --write
```

Tal y como se menciona en la última línea impresa habiendo hecho

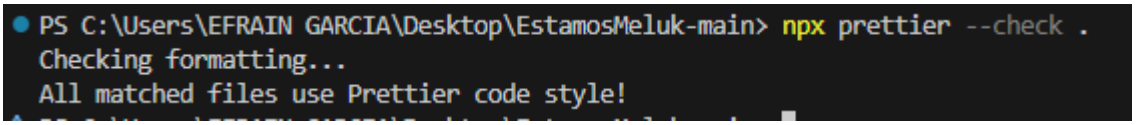
```
JavaScript
npx prettier --check .
```

Al ejecutar el comando se muestra lo siguiente:



De donde se puede deducir que las líneas donde no se encuentra ningún paréntesis son aquellos archivos que contenían errores según el estilo de código Prettier y, por tanto, aquellas que fueron editadas.

Se vuelve a ejecutar el comando ‘npx prettier --check .’ y ahora este imprime:

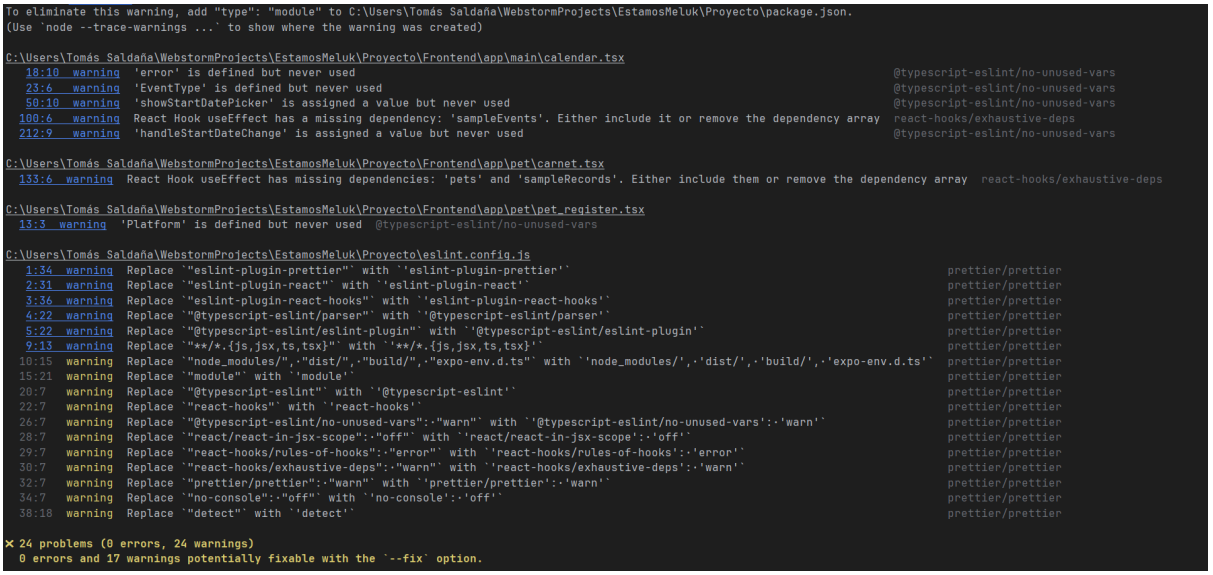


Mostrando que ahora los archivos que contenían errores fueron editados de manera correcta, como se quería ver.

Eslint

Con Eslint también se realizó la corrección de errores usando los siguientes comandos.
Para Analizar errores

```
JavaScript
npx eslint . --ext .js,.ts,.tsx
```



Después de ejecutar npx eslint . Se identificaron diversos problemas en el proyecto.
La mayoría no son fallos graves, sino advertencias relacionadas con **estilo, buenas prácticas y configuración**.

Variables declaradas pero no utilizadas

Ejemplos detectados:

- error declarado pero nunca usado.
- EventType declarado pero nunca usado.
- Platform importado pero no utilizado.

Son valores o imports presentes en el archivo pero no utilizados en el código, por lo que ESLint advierte que pueden eliminarse para mantener el código limpio.

2. Hooks de React con dependencias faltantes

Ejemplos:

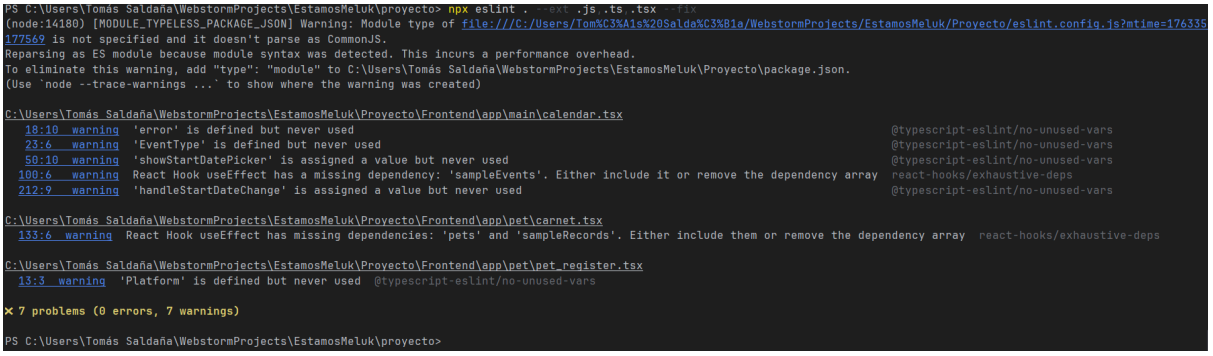
- useEffect sin sampleEvents
- useEffect sin pets o sampleRecords

React recomienda declarar todas las dependencias usadas dentro del useEffect.
Si no se hace, puede causar comportamientos inesperados, por ejemplo:

- no actualizar la UI correctamente
- estados desincronizados

y para arreglarlos arrojo lo siguiente al usar este comando:

```
JavaScript
npx eslint --fix
```



Donde básicamente corrigió los errores de variables no usadas y pasó todo a código más limpio.