Análisis de código React con ESLint

Tiempo estimado: 10min

Antes de finalizar el curso, vamos a presentar cómo comprobar si el código de nuestra aplicación React cumple la sintaxis de JavaScript y JSX. Esto se hace mediante herramientas de análisis de código como, por ejemplo, ESLint. En esta lección, no describimos ESLint, sino cómo usarlo para que pueda analizar el código de una aplicación React.

Comenzamos recordando muy brevemente en qué consiste el análisis de código estático y por qué se utiliza ESLint, en vez de otras herramientas. Y a continuación, se presenta el *plugin* eslint-plugin-react con el que poder aplicar reglas específicas de JSX y React.

Al finalizar la lección, el estudiante:

- Sabrá cómo utilizar ESLint para analizar aplicaciones React.

Introducción

ESLint es una herramienta de análisis de código estático (*static code analysis*) que lee código, sin llegar a ejecutarlo, con objeto de detectar errores y patrones poco recomendables. Lo que se busca es que el código cumpla las reglas sintácticas de JavaScript, además de los estándares o convenciones definidos por la organización.

Formalmente, este tipo de herramientas se conocen como *linters*. Y en JavaScript, podemos encontrar varias como, por ejemplo, JSHint, JSLint y ESLint. ESLint es capaz de analizar código JavaScript, JSX y comprobar reglas específicas de React. Otros sólo son capaces de analizar código JavaScript.

La versión de fábrica de ESLint no es capaz de analizar JSX y React. Pero como se puede añadir nueva funcionalidad a ESLint muy fácilmente, es posible añadirle este soporte. Esto se consigue mediante el *plugin* eslint-plugin-react. Recordemos que un *plugin* no es más que un componente que añade nuevas reglas a ESLint, permitiéndonos así extender su funcionalidad.

Plugin eslint-plugin-react

Para que ESLint pueda analizar código JSX y React, hay que configurar el *plugin* eslint-plugin-react. Esta configuración consiste en:

1. Añadir el *plugin* a las dependencias del proyecto.
2. Indicarle a ESLint que use el *plugin*.
3. Configurar las reglas del *plugin* a aplicar.

Añadidura del plugin a las dependencias del proyecto

Lo primero es añadir el *plugin* a las dependencias de desarrollo del proyecto. Recordemos, propiedad devDependencies del archivo package.json:

```
"devDependencies": {
  "eslint-plugin-react": "*"
}
```

Uso del plugin

A continuación, hay que indicarle a ESLint que utilice el *plugin*. Consiste en dos pasos. Por un lado, registrar el *plugin* en el archivo .eslintrc mediante la propiedad plugins:

```
"plugins": [
  "react"
]
```

Por otra parte, y teniendo en cuenta que de fábrica ESLint sólo analiza código JavaScript, hay que indicarle que también analice JSX. En el mismo archivo .eslintrc, configurar la propiedad jsx a true, en la

propiedad parserOptions.ecmaFeatures:

```
"parserOptions": {
  "ecmaFeatures": {
    "jsx": true
  }
}
```

Reglas del plugin

Una vez configurado el *plugin*, lo siguiente es configurar sus reglas. Recordemos que mediante la propiedad extends del archivo .eslintrc podemos aplicar una configuración predeterminada de reglas. Así por ejemplo, mediante eslint:recommended se aplicarán las reglas predeterminadas recomendadas según la configuración del fabricante. Con el *plugin* de React, también podemos usar una configuración de fábrica. Ejemplo:

```
"extends": [
  "eslint:recommended",
  "plugin:react/recommended"
]
```

Si lo deseamos, podemos configurar que se aplique todas las reglas mediante plugin:react/all.

Reglas relacionadas con JSX

En primer lugar, vamos a presentar las reglas JSX más utilizadas, la lista completa se encuentra en github.com/yannickcr/eslint-plugin-react#jsx-specific-rules:

Regla	Tipo	Descripción
react/jsx-boolean-value	Obligación	Configura cómo debe indicarse el valor de las propiedades booleanas: • always. Siempre debe indicarse, por ejemplo, mediante {true}. No se puede indicar propiedades sin valor como, por ejemplo, disabled. • never. Nunca debe indicarse el valor true. Sólo se debe indicar el nombre.
react/jsx-filename-extension	Obligación	Configura cuál debe ser la extensión de los archivos que contienen código JSX. Se indica mediante el segundo elemento de la regla, mediante un objeto con la propiedad extensions. Ejemplo: "react/jsx-filename-extension": ["error", {"extensions": [".js", ".jsx"]}].
react/jsx-indent	Obligación	Configura cómo debe ser el sangrado del código JSX. Similar a la regla indent de ESLint para JavaScript, pero aplicada a JSX. Su valor puede ser tab o un número que indica el número de espacios.
react/jsx-key	Obligación	Configura si los elementos iterables tienen la propiedad key.
react/jsx-no-bind	Prohibición	Configura si es posible usar bind() o funciones de flecha en la expresión valor de una propiedad. Se utiliza un objeto con las siguientes propiedades: • allowArrowFunctions (boolean). Se puede indicar funciones de flecha. • allowBind (boolean). Se puede usar bind().
react/jsx-no-duplicate-props	Prohibición	Prohíbe la duplicidad de propiedades en el mismo elemento.
react/jsx-no-target-blank	Prohibición	Prohíbe fijar la propiedad target a _blank.
react/jsx-no-undef	Prohibición	Prohíbe usar variables no declaradas.

react/jsx-pascal-case	Obligación	Requiere que los nombres de componentes se definan en formato *PascalCase*.
react/jsx-space-before-closing	Obligación	Requiere que los elementos que se autocierran tengan un espacio antes de />.
react/jsx-uses-react	Obligación	Requiere que se importe React en todo archivo que defina un componente, impidiendo que react/jsx-no-undef propague error aun si no se usa el objeto.
react/jsx-wrap-multilines	Obligación	Obliga a que un elemento que se expande en múltiples líneas se delimite entre paréntesis.

Reglas relacionas con React

Las reglas específicas de React más comunes son:

Regla	**Tipo**	**Descripción**
react/display-name	Obligación	Todo componente debe definir la propiedad displayName.
react/no-deprecated	Prohibición	Impide el uso de métodos deprecados.
react/no-did-mount-set-state	Prohibición	Impide fijar el estado, setState(), en el método de ciclo de vida componentDidMount().
react/no-did-update-set-state	Prohibición	Impide fijar el estado, setState(), en el método de ciclo de vida componentDidUpdate().
react/no-direct-mutation-state	Prohibición	Impide modificar el estado directamente sobre la propiedad state.
react/no-is-mounted	Prohibición	Impide el uso del método isMounted().
react/no-multi-comp	Prohibición	Impide definir varios componentes en el mismo archivo.
react/no-set-state	Prohibición	Impide el uso de setState(). Generalmente, se usa en aplicaciones de sólo lectura donde todos los componentes son inmutables.
react/no-string-refs	Prohibición	Impide asignar un valor de tipo texto a la propiedad ref.
react/prefer-es6-class	Obligación	Los componentes deben definirse mediante ES6, en vez de mediante React.createClass().
react/prop-types	Obligación	Obliga a definir la propiedad propTypes.
react/require-optimization	Obligación	Obliga a que todo componente defina el método de ciclo de vida shouldComponentUpdate().
react/style-prop-object	Obligación	Obliga a que la propiedad style siempre reciba como valor un objeto.

La lista completa se encuentra en github.com/yannickcr/eslint-plugin-react#list-of-supported-rules.

Generador de Justo

Si utilizamos el generador de Justo para crear la estructura inicial del proyecto React, se creará una configuración necesaria y, en muchas ocasiones, suficiente de ESLint. Registra el *plugin* en los archivos package.json y .eslintrc e indica la configuración predeterminada a aplicar con JSX y React.

