Vite, Prettier y ESLint

Nota: Las extensiones ESLint y Prettier deben estar instaladas en VS Code.

Para crear un proyecto de React con Vite:

```
npm create vite@latest

npm create vite@latest .

npm create vite@latest my-project-name -- --template react

npm create vite@latest my-project-name -- --template react-ts

npm i -D vite @vitejs/plugin-react
```

Si elegimos la primera opción, nos pedirá el nombre del proyecto y se creará en una carpeta nueva con el nombre del proyecto. Si ponemos `.` como nombre del proyecto, se creará en el directorio actual sin hacer una carpeta nueva y el nombre del proyecto será el de la carpeta actual.

Luego nos darán a elegir un framework:
`Vanilla` | `Vue` | `React` | `Preact` | `Lit` | `Svelte` | `Solid` | `Qwik` | `Others`
Si elegimos `React` después elegiremos entre JavaScript o TypeScript.

Esto crea un proyecto con React y ESLint, pero Prettier hay que instalarlo por separado (ver más adelante).
Por último, habrá que instalar los paquetes con `npm i` desde la carpeta del proyecto recién creado.

Para ejecutar live server:	`npm run dev`
Para acceder desde otro dispositivo:	`npm run dev -- --host`
Para hacer un build:	`npm run build`
Para ver el build:	`npm run preview`

¿Para qué sirven ESLint y Prettier?

ESLint es un linter, Prettier es un formateador.
Un linter entiende código y por eso es capaz de señalar errores y forzar reglas.
Un formateador sólo se ocupa de la presentación de ese código.
VS Code ya tiene preconfigurado un linter y un formateador, por eso el código se formatea aunque no estén instalados ES Lint ni Prettier.
Es recomendable instalar otros (como ESLint y Prettier) porque facilita el uso de reglas comunes entre varios usuarios. Pero como a veces sus funciones se solapan, hay que configurarlos para que se lleven bien.

Prettier

Prettier no se instala cuando creamos un proyecto con Vite o Next.js, así que hay que hacerlo manualmente con el siguiente comando:

```
npm i -D prettier

npm i -D --save-exact prettier       // Impide actualizaciones
```

Plugin eslint-config-prettier

Para que ESLint y Prettier no entren en conflicto instalamos también el plugin `eslint-config-prettier`:

```
npm i -D eslint-config-prettier
```

Y añadimos `"prettier"` al final del array `extends` en el archivo de configuración de ESLint. Este `"prettier"` se refiere al plugin que acabamos de instalar y siempre debe estar **AL FINAL** del array:

.eslintrc.json

```
"extends": [
  "eslint:recommended",
  "plugin:react/recommended",
  "prettier"                                        // Siempre el último
]
```

Configuración de Prettier

Para cambiar los ajustes por defecto de Prettier tenemos que crear el archivo `.prettierrc.json` o sencillamente `.prettierrc`, sin extensión. Es posible que ya exista el archivo `.prettierrc` y contenga un objeto vacío: `{}` Esto básicamente quiere decir que Prettier está usando los ajustes por defecto. Y también es una manera de obligar a VS Code a usar Prettier automáticamente.
En este archivo podemos cambiar los ajustes por defecto. Por ejemplo:

.prettierrc

```
{
  "semi": false,              ← Sin ';' al final de las líneas
  "singleQuote": true         ← 'Comillas sencillas' en lugar de "dobles"
  "arrowParens": "avoid",     ← Omite paréntesis en funciones de flecha si es posible
  "endOfLine": "lf",          ← Salto de línea: lf | crlf | cr | auto
  "jsxSingleQuote": true,
  "tabWidth": 2,
  "trailingComma": "none",    ← Probar también "es5" o "all"
  "proseWrap": "always",      ← Probar también "never" o "preserve"
  "printWidth": 80,
  "plugins": ["prettier-plugin-tailwindcss"],
  "useTabs": false,           ← Por defecto indenta con espacios, no con tabuladores
}
```

Más opciones: https://prettier.io/docs/en/options

Ajustes de Prettier en VS Code

Atención a los ajustes de VS Code: En el campo `Prettier: Require Config` se puede marcar si se requiere un archivo de configuración para formatear con Prettier. La ventaja de tener estos archivos de configuración

de Prettier y VS Code es que el proyecto siempre tendrá los mismos ajustes independientemente de los ajustes de VS Code de cada usuario.

También podemos añadir un archivo `settings.json` en la carpeta `.vscode` del proyecto:

.vscode/settings.json

```
{
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.formatOnSave": true
}
```

ESLint

Archivo de configuración

El archivo de configuración de ESLint se llama `.eslintrc` y lo podemos encontrar en el directorio raíz con distintas extensiones: `.js`, `.cjs`, `.yaml`, `.yml`, `.json`.
A partir de la versión 9.x se usan archivos `eslint.config` con extensiones `.js`, `.mjs` o `.cjs`.

Error Prop Types

En el archivo de configuración de ESLint añadimos la siguiente regla al objeto `rules`:

.eslintrc.cjs

```
rules: {
  'react/prop-types': 'off'
}
```

Error module.exports

Si `module.exports` da fallos, se puede arreglar añadiendo `node: true` al objeto `env`:

.eslintrc.cjs

```
env: {
  browser: true,
  es2020: true,
  node: true
}
```

Error si no se importa React

Para que no salga error si omitimos `import React from 'react'` añadimos la siguiente regla:

.eslintrc.cjs

```
rules: {
  'react/react-in-jsx-scope': 'off'
}
```

De todas maneras, este error se puede evitar si añadimos este plugin en `extends`:

.eslintrc.cjs

```
extends: [
  'plugin:react/jsx-runtime'
]
```

Logical assignment operators

Muestra un error si una expresión se podría resumir con un operador de asignación.
Por ejemplo, `count ||= 0` en lugar de `count = count || 0`.

```
rules: {
  'logical-assignment-operators': 'error'
}
```

Instalación manual de ESLint

ESLint ya se instala y se configura con plugins de React si iniciamos un proyecto con Vite o Next.js.
De todas formas, si hiciera falta instalarlo manualmente hay dos opciones:

```
npm init @eslint/config       // también npx eslint --init

npm install -D eslint
```

Si elegimos la primera opción, nos preguntará para qué queremos ESLint:

```
  To check syntax only
> To check syntax and find problems
  To check syntax, find problems and enforce code style
```

Elegimos la segunda, puesto que para forzar un estilo de código usaremos Prettier.

Después nos preguntará qué tipo de módulos de JavaScript usaremos:

```
> JavaScript modules (import/export)
  CommonJS (require/exports)
  None of these
```

Framework: `React`, `Vue.js`, `None of these`
TypeScript: `No`, `Yes`
Dónde se ejecutará el código: `Browser`, `Node` (admite selección múltiple)
Guía de estilo: `Airbnb`, `Standard`, `Google`, `XO`
Formato de archivo config: `JavaScript`, `YAML`, `JSON` (mejor elegir `JSON`)

Tutorial Brian Holt

En `.eslintrc.json`, Brian Holt añade las siguientes líneas:

.eslintrc.json

```
"parserOptions": {
  "ecmaVersion": "latest",
  "sourceType": "module"
  "ecmaFeatures": {
    "jsx": true
  }
},
```

También añade unos plugins para que ESLint entienda algunas peculiaridades de JSX (es posible que ya se haya solucionado desde entonces y tal vez no haga falta añadir nada):

```
npm i -D eslint-plugin-import
npm i -D eslint-plugin-jsx-a11y      ← a11y se escribe con unos (11), no con eles (ll)
npm i -D eslint-plugin-react
```

(En realidad todo esto se pude combinar en una sola línea, pero prefiero tenerlo por separado).
Estas dependencias deben figurar en `.eslintrc.json`, dentro de `"extends"`. `"prettier"` siempre al final:

```
"extends": [
  "eslint:recommended",
  "plugin:import/errors",
  "plugin:react/recommended",
  "plugin:jsx-a11y/recommended",
  "prettier"
],
```

Y también se añaden en plugins:

```
"plugins": [
  "react",
  "import",
  "jsx-a11y"
]
```

Para que no salte el error de Prop Types, añadimos `"rules"` al objeto:

```
"rules": {
  "react/prop-types": 0,
  "react/react-in-jsx-scope": 0
},
```

(No sé si es `"0"` u `"off"`)

Por último, Brian Holt añade lo siguiente:

```
  "settings": {
    "react": {
      "version": "detect"
    },
    "import/resolver": {
      "node": {
        "extensions": [".js", ".jsx"]
      }
```

```
    }
  }
```

Configuración final según Brian Holt:

.eslintrc.json

```
{
  "extends": [
    "eslint:recommended",
    "plugin:import/errors",
    "plugin:react/recommended",
    "plugin:jsx-a11y/recommended",
    "prettier"
  ],
  "rules": {
    "react/prop-types": 0,
    "react/react-in-jsx-scope": 0
  },
  "plugins": ["react", "import", "jsx-a11y"],
  "parserOptions": {
    "ecmaVersion": 2022,
    "sourceType": "module",
    "ecmaFeatures": {
      "jsx": true
    }
  },
  "env": {
    "es6": true,
    "browser": true,
    "node": true
  },
  "settings": {
    "react": {
      "version": "detect"
    },
    "import/resolver": {
      "node": {
        "extensions": [".js", ".jsx"]
      }
    }
  }
}
```