Manual de estándares y buenas practicas a seguir en el equipo de desarrollo

Version: *1.0.0* Elaborado por: *Andrés Escala*

Para mantener una buena estructura de los diferentes componentes del proyecto, siguiendo las buenas prácticas propuestas por la comunidad de desarroladores, se adoptarán los siguientes estándares y buenas prácticas de tecnología:

Estándares:

1. ECMAScript 2021 (ES12)
2. JS Coding Standards: https://github.com/0granada/js-coding-standards

Nombre de archivos:

Utiliza nombres descriptivos que indiquen claramente el contenido del archivo. Utiliza minúsculas y guiones bajos para separar palabras en lugar de espacios o mayúsculas. Evita utilizar caracteres especiales o acentos en los nombres de los archivos. Utiliza la extensión de archivo adecuada para el tipo de archivo (por ejemplo, .html para archivos HTML, .css para archivos CSS, .js para archivos JavaScript, etc.).

Nombrar Variables

- Utiliza nombres descriptivos que indiquen claramente el propósito de la variable.
- Utiliza camelCase (la primera letra de la primera palabra en minúscula y la primera letra de las siguientes palabras en mayúscula) para nombrar las variables.
- Evita utilizar nombres demasiado largos o abreviaturas confusas.
- Utiliza nombres coherentes y consistentes en todo el proyecto.
- Utiliza nombres que no sean palabras reservadas en JavaScript. Ej: ***estaDisponible***
  ** Si la variable es constante, utiliza todo en mayúsculas y guiones bajos para separar palabras en lugar de espacios o mayúsculas.* ***MAX_NUMERO_LIBROS***

Nombres de funciones en JavaScript

- Utiliza nombres descriptivos que indiquen claramente lo que hace la función. Utiliza **camelCase** (la primera letra de la primera palabra en minúscula y la primera letra de las siguientes palabras en mayúscula) para nombrar las funciones.
- Evita utilizar nombres demasiado largos o abreviaturas confusas y/o nombre genéricos que no sean necesarios, como por ejemplo: **script.js** o **main.js**.
- Si la función es un constructor, utiliza PascalCase (la primera letra de cada palabra en mayúscula). Nombres de archivos:

Nombres de clases CSS

- Utiliza nombres descriptivos que indiquen claramente el propósito de la clase.
- Utiliza minúsculas y guiones bajos para separar palabras en lugar de espacios o mayúsculas.
- Evita utilizar nombres demasiado largos o abreviaturas confusas.
- Utiliza nombres coherentes y consistentes en todo el proyecto.

Nombres de archivos HTML

- Utiliza nombres descriptivos que indiquen claramente el contenido del archivo.
- Utiliza minúsculas y guiones bajos para separar palabras en lugar de espacios o mayúsculas.
- Evita utilizar caracteres especiales o acentos en los nombres de los archivos.
- Utiliza nombres coherentes y consistentes en todo el proyecto.

React.JS

- Estructura de archivos: se recomienda organizar los archivos de una aplicación React en una estructura de carpetas coherente y fácil de seguir, separando los componentes, estilos y otros archivos en carpetas separadas.
- Convenciones de nomenclatura: es recomendable utilizar nombres descriptivos y coherentes para los componentes y otros elementos de la aplicación, utilizando PascalCase para los nombres de componentes y camelCase para las propiedades y variables.
- Componentes funcionales: se recomienda utilizar componentes funcionales siempre que sea posible, ya que son más fáciles de leer y mantener que los componentes de clase.
- Separación de responsabilidades: es importante separar las responsabilidades de los componentes, evitando que un componente tenga demasiadas tareas o dependencias.
- Uso de propiedades: se recomienda utilizar las propiedades (props) para pasar datos entre componentes, evitando el uso de variables globales o estados compartidos.
- Uso de estados: se recomienda utilizar los estados (state) para gestionar los datos que cambian con el tiempo en la aplicación, evitando el uso excesivo de estados complejos o anidados.
- Uso de hooks: es recomendable utilizar los hooks de React (como useState y useEffect) para gestionar el estado y el ciclo de vida de los componentes, en lugar de utilizar métodos del ciclo de vida de los componentes de clase.
- Utilizar nombrado de componentes en PascalCase y propiedades en camelCase.
- Utilizar JSX para escribir código HTML dentro de JavaScript.
- Utilizar el método "render" para devolver el contenido del componente.