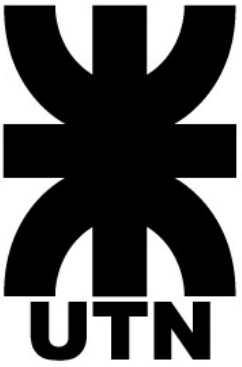
UNIVERSIDAD TECNOLÓGICA NACIONAL
Facultad Regional Córdoba



Ingeniería en Sistemas de Información

Ingeniería y Calidad de Software

TP evaluable N°6

4K3 - Grupo 12

Docentes:

- Ing. Laura Covaro - lcovaro@gmail.com
- Ing. Cecilia Massano - ceciliamassano@gmail.com
- Ing. Georgina González - gg.georginagonzalez@gmail.com

Integrantes:

Apellido y Nombre	Legajo	Correo electrónico
Arias Felipe	82412	felipe2arias@hotmail.com
Benedetti Angelo	81699	angelo.benedetti2910@gmail.com
Bessone Luca Sebastián	82661	bessonelucasebastian@gmail.com
Cantero Santiago	81658	santicantero124@gmail.com
Miszczuk Solange	58535	solangedbmiszczuk@gmail.com
Morales Ortiz Paula Eileen	79958	paula.morales.utn@gmail.com
Moriconi Lorenzo	79478	7moriconi@gmail.com
Soldera Federico	79554	fedesoldera@gmail.com

Año: 2024

Documento de Buenas Prácticas y Reglas de Estilo de Código para Proyectos en React

Convenciones Generales

1. **Indentación y Espaciado:**
    - Utilizar una indentación de 2 espacios.
    - No utilizar tabulaciones para la indentación.
    - Mantener una línea en blanco al final de cada archivo.
    - Dejar un espacio después de cerrar cada condicional.

2. **Nombres de Variables y Funciones:**
    - Utilizar camelCase para nombrar variables y funciones.
    - Utilizar nombres descriptivos y significativos.
    - Evitar nombres de una sola letra, a menos que sea un caso muy específico.
3. **Componentes:**
    - Nombrar los componentes con nombres descriptivos en mayúscula inicial.
    - Utilizar fragmentos o elementos contenedores para envolver múltiples elementos JSX.
    - Separar los componentes en archivos individuales.

Estilo de Código

1. **Declaración de Variables:**

    - Utiliza “ **var** ” siempre que sea posible y cuando el compilador lo admita.
    - Utilizar “ **const** ”para declarar variables que no se reasignan.
    - Siempre las variables deben comenzar con minúscula.

- Los nombres de variables, propiedades y métodos deben ser claros y descriptivos, de modo que transmitan su propósito a quienes revisen el código. Se permite el uso de abreviaturas como "**div**" en vez de "**división**".
- Evitar evitar que los nombres sean muy largos, porque complica la lectura del código.
- Evitar crear variables globales innecesarias.
- Evitar crear variables de un solo uso, porque ocupan memoria de manera innecesaria.

2. **Estructura del repositorio :**

- Usar un archivo *.gitignore* apropiado:
  - Asegúrate de incluir un archivo *.gitignore* con las configuraciones necesarias para ignorar archivos y carpetas no deseados,
  - como **node_modules**, **.env**, y los archivos de construcción como **build/** o **dist/**.
- Incluir un archivo *README.md* detallado:
  - Proporciona información clara sobre el propósito del proyecto
  - Cómo configurarlo,
  - Instrucciones de instalación,
  - scripts útiles y
  - Cómo contribuir.
  - Esto ayuda a futuros colaboradores y facilita la comprensión del proyecto.
- Gestionar las dependencias de manera eficiente:
  - Utiliza versiones específicas de las dependencias para asegurar la estabilidad del proyecto.
  - Incluye un archivo *package-lock.json* o *yarn.lock* para mantener la integridad de las dependencias y evitar posibles conflictos entre versiones.
- Estructura de Carpetas Clara y Escalable:
  - Organiza el código con una estructura lógica, separando componentes, estilos, utilidades y servicios.
  - Por ejemplo, puedes seguir la estructura básica:

```
src/
    components/
    hooks/
    utils/
    styles/
    services/
```

Buenas Prácticas

1. **División de Responsabilidades:**
    - Componentes Pequeños y Reutilizables:
        - Descompón la interfaz en componentes pequeños y especializados que hagan una única tarea.
        - Esto facilita su reutilización y mantenimiento.
        - Evita crear componentes monolíticos que manejen demasiadas responsabilidades a la vez.
    - Separación de Lógica y Presentación:
        - Divide los componentes en "contenedores" (smart components) y componentes de presentación (dumb components).
        - Los contenedores manejan la lógica de negocio (estado, lógica de datos), mientras que los componentes de presentación solo se encargan de la interfaz y recibir props.
    - Uso Adecuado de Hooks:
        - Aprovecha los hooks de React para encapsular la lógica reutilizable en funciones separadas.
        - Por ejemplo, crea hooks personalizados (useFetch, useForm) para mantener la lógica fuera de los componentes de presentación y mejorar la legibilidad.
    - Contexto y Reducers para Manejo de Estado Global:
        - Usa **React Context** o bibliotecas como **Redux** para manejar el estado compartido entre múltiples componentes.
        - Mantén la lógica de gestión de estado separada del resto de la aplicación, para que los componentes no se ocupen de gestionar datos globales directamente.
    - División de Responsabilidades en Funciones y Utilidades:
        - Extrae lógica compleja o reutilizable fuera de los componentes en archivos de utilidad (*utils/*) o servicios (*services/*).
        - Por ejemplo, lógica relacionada con llamadas a APIs o formateo de datos debería residir en funciones específicas fuera del componente, para mantenerlos más simples y legibles.

2. **Manejo de Errores:**
    - Componentes de Error Boundary:
        - Utiliza Error Boundaries para capturar y manejar errores inesperados en la interfaz de usuario.
        - Un componente de tipo Error Boundary puede envolver partes específicas de tu aplicación y prevenir que un error en una parte de la UI bloquee toda la aplicación.
        - Se implementa usando el método **componentDidCatch** o las APIs modernas de hooks.
    - Manejo de Errores en Promesas y Fetch:
        - Siempre maneja errores al realizar solicitudes asíncronas, como llamadas a API usando fetch, axios, etc.
        - Implementa bloques **try/catch** o utiliza **catch()** en promesas para manejar respuestas fallidas o errores de red.
    - Uso de Mensajes de Error Informativos:
        - Asegúrate de proporcionar mensajes de error claros y

específicos para los usuarios.
        - Cuando ocurra un error, muestra un mensaje útil en lugar de una pantalla en blanco o un mensaje genérico.
        - Esto mejora la experiencia del usuario y facilita la depuración.
    - Manejo Centralizado de Errores:
        - Implementa un sistema centralizado para manejar y registrar errores, especialmente en aplicaciones grandes.
        - Puedes utilizar servicios como **Sentry** o **LogRocket** para monitorear y registrar errores que ocurren en producción, lo que facilita la identificación y corrección de problemas antes de que afecten a m ás usuarios.

3. **Utilización de Tests**
    - Usa **React Testing Library** para Pruebas de Componentes:
        - **React Testing Library (RTL)** es la herramienta recomendada para probar componentes en React.
        - En lugar de enfocarse en los detalles de implementación como lo hace **Enzyme**, **RTL** se centra en cómo interactúan los usuarios con la aplicación, lo que resulta en pruebas más robustas y orientadas al comportamiento real de la UI
    - Escribe Pruebas Unitarias y de Integración:
        - Asegúrate de incluir tanto pruebas unitarias (para componentes pequeños o funciones) como pruebas de integración (para comprobar cómo interactúan múltiples componentes entre sí).
        - Las pruebas unitarias verifican que cada parte funcione correctamente de forma aislada, mientras que las pruebas de integración aseguran que las diferentes partes colaboren como se espera.