Taller 5 – Patrones

1. Información General del proyecto

   Desarrollar un semiclone de la web de MercadoLibre conlleva una serie de desafíos y aprendizajes significativos en el ámbito del desarrollo web. Este proyecto se plantea con el propósito principal de proporcionar a los desarrolladores una experiencia práctica en el consumo de API's externas y en la creación del frontend utilizando la biblioteca React.JS.

   - **Propósito del Proyecto:** El objetivo fundamental de este proyecto es brindar a los desarrolladores una oportunidad práctica para aprender y aplicar conceptos esenciales en dos áreas cruciales del desarrollo web: la integración con API's externas y la construcción de interfaces de usuario interactivas con React.JS. Al replicar la funcionalidad principal de MercadoLibre, los participantes podrán familiarizarse con el flujo de trabajo necesario para consumir datos de una API, así como comprender cómo estructurar y diseñar componentes frontales utilizando React.

   - **Estructura General del Diseño:** El proyecto se organizará en dos partes principales: el back-end que se encargará de manejar las peticiones a la API de MercadoLibre y proporcionar los datos necesarios, y el front-end que será construido con React.JS para crear una interfaz de usuario similar a la de MercadoLibre. La estructura general del diseño podría incluir los siguientes componentes:

     1. **Backend:**

        - **Servidor/API:** Encargado de manejar las solicitudes del cliente y comunicarse con la API de MercadoLibre para obtener información sobre productos, precios, etc.

        - **Endpoints:** Rutas específicas para obtener datos de productos, detalles de productos, imágenes, etc.

        - **Manejo de Errores y Excepciones:** Implementación de un manejo adecuado de errores para garantizar una experiencia fluida.

     2. **Frontend:**

        - **Componentes React:** Desarrollo de componentes reutilizables para la creación de la interfaz de usuario, como la barra de búsqueda, la lista de productos, los detalles del producto, etc.

        - **Enrutamiento:** Implementación de un sistema de enrutamiento para navegar entre las diferentes secciones de la aplicación.

- **Gestión de Estado:** Uso de estados y propiedades para gestionar la información y la interacción del usuario.

- **Integración de Imágenes:** Manejo de imágenes de productos obtenidas de la API.

- **Grandes Retos de Diseño:** Este proyecto enfrentará varios retos de diseño, entre los cuales destacan:

    1. **Consumo de API Externa:** La integración exitosa con la API de MercadoLibre implica comprender la autenticación, la estructura de los endpoints y cómo manejar la respuesta de la API.

    2. **Diseño Responsivo:** Garantizar que la aplicación sea accesible y atractiva en diferentes dispositivos, desde computadoras de escritorio hasta dispositivos móviles.

    3. **Gestión de Estado Eficiente:** Utilizar el estado de manera eficiente en la aplicación React para garantizar una experiencia de usuario fluida y sin errores.

    4. **Manejo de Errores:** Implementar un manejo robusto de errores tanto en el backend como en el frontend para brindar una experiencia de usuario sólida incluso en situaciones inesperadas.

    5. **Optimización de Imágenes y Carga Rápida:** Asegurar que las imágenes se carguen de manera eficiente para evitar tiempos de carga prolongados.

En conclusión, desarrollar un semiclone de la web de MercadoLibre no solo implica replicar la funcionalidad superficial, sino también comprender y superar los desafíos asociados con la integración de API's externas, el diseño reactivo y la gestión eficiente del estado en React. Este proyecto proporciona a los desarrolladores una experiencia práctica invaluable para fortalecer sus habilidades en el desarrollo web full-stack.

2. Patrón GoF

Luego de examinar el proyecto y comprender sus requisitos funcionales, se evidencia la aplicación del patrón "Singleton" para gestionar los objetos "institutions_existence" y "authors_existence". Estos funcionan como listas únicas destinadas a almacenar las instituciones y autores ya existentes. Dichas listas desempeñan un papel crucial al verificar la existencia previa de una institución o autor antes de crear una nueva instancia. En caso de existir, se actualizan sus atributos en lugar de generar una instancia adicional. Esta práctica no solo evita la creación redundante de múltiples instancias de la misma institución o autor, sino que también asegura que se manipule la misma instancia

cuando ya está presente en las listas. Este enfoque contribuye a una gestión más eficiente y coherente de los datos existentes en el sistema.

La aplicación de este patrón presenta tanto beneficios como limitaciones.

Ventajas de Utilizar el Patrón:

1. **Optimización de Recursos:** Evita la generación innecesaria de instancias, optimizando el consumo de recursos.

2. **Uniformidad de Datos:** Asegura la coherencia al trabajar con la misma instancia, manteniendo la uniformidad de los datos.

3. **Acceso Global:** Ofrece un único punto de entrada global para las listas, simplificando su gestión.

Desventajas de Utilizar el Patrón:

1. **Restricción de Flexibilidad:** Podría limitar la flexibilidad en contextos donde se requieren múltiples instancias.

Alternativas Considerables:

1. **Gestión Manual de Instancias:** En lugar de depender exclusivamente del patrón “Singleton”, se podría gestionar de forma manual la creación y actualización de instancias según la necesidad.

2. **Empleo de Estructuras Especializadas:** Utilizar estructuras de datos específicas, como diccionarios, para administrar entidades únicas sin depender del patrón “Singleton”.

En resumen, la aplicación del patrón “Singleton” en este proyecto proporciona una solución eficaz para garantizar coherencia y eficiencia en la manipulación de datos. A pesar de sus indiscutibles ventajas, es crucial sopesar las posibles limitaciones y evaluar alternativas conforme a los requisitos particulares del proyecto.

```
for c in _authors:
    if c not in authors_existence:
        authors_existence += [c]
        authors += [Author(c, _institutions, _fields, _article)]
    else:
        pos = authors_existence.index(c)

        authors[pos].articles += [_article]

        for field in _fields:
            if field not in authors[pos].fields:
                authors[pos].fields += [field]

        for institution in _institutions:
            if institution not in authors[pos].institutions:
                authors[pos].institutions += [institution]

        if _article not in authors[pos].articles:
            authors[pos].articles += _article
```

```
        for insti in _institutions:
            if insti not in institutions_existence:
                institutions_existence += [insti]
                institutions += [Institution(insti, _authors, _fields, [_article])]
            else:
                pos = institutions_existence.index(insti)

                for author in _authors:
                    if author not in institutions[pos].author:
                        institutions[pos].author += [author]

                for field in _fields:
                    if field not in institutions[pos].fields:
                        institutions[pos].fields += [field]

                if _article not in institutions[pos].articles:
                    institutions[pos].articles += [_article]

```