UNIVERSIDAD DON BOSCO

Principios Solid

Integrantes:

- José Luis Bardales Paniagua BP171968
- Diego Alberto Vásquez Arias VA172033

Asignatura:

Diseño y Programación de Software Multiplataforma

Docente:

Ing. Alexander Sigüenza

Fecha de entrega:

10 de junio de 2023

Contenido

Introducción

Los principios SOLID son un conjunto de directrices de diseño de software que nos ayudan a crear aplicaciones sólidas, escalables y mantenibles. Estos principios son fundamentales en el desarrollo de software en general, incluyendo el desarrollo de aplicaciones en React Native.

Los principios SOLID son un conjunto de directrices de diseño de software que nos ayudan a crear aplicaciones sólidas, escalables y mantenibles. Estos principios son fundamentales en el desarrollo de software en general, incluyendo el desarrollo de aplicaciones en React Native.

El ejemplo del componente TarjetaProducto en React Native ilustra cómo aplicar el principio de Abierto/Cerrado. Al separar las responsabilidades de mostrar y editar en componentes diferentes, podemos ampliar el comportamiento sin modificar el código original, mejorando así la legibilidad y el mantenimiento del código.

Al aplicar estos principios SOLID en React Native, podemos crear aplicaciones sólidas, escalables y mantenibles. Sin embargo, es importante adaptar estos principios a las necesidades específicas de cada proyecto y contexto.

Principios SOLID

1. Principio de Responsabilidad Única

   Este principio establece que cada componente debe tener un propósito claro y específico. En React Native, esto significa que cada componente debe encargarse de una sola tarea o función particular. Esto hace que los componentes sean más comprensibles, fáciles de probar y mantener. Además, permite la reutilización eficiente de componentes en diferentes partes de la aplicación.

2. Principio de Abierto/Cerrado

   Este principio sugiere que los componentes deben ser abiertos a la extensión, pero cerrados a la modificación. En React Native, esto se traduce en diseñar componentes de manera que puedan ser ampliados y personalizados sin necesidad de alterar el código original. Para lograrlo, se pueden utilizar propiedades (props) para configurar el comportamiento del componente.

3. Principio de Sustitución de Liskov

   Este principio establece que los objetos derivados deben poder ser utilizados en lugar de los objetos de la clase base sin causar errores o problemas. En el contexto de React Native, esto implica que cualquier componente que herede de otro componente debe ser completamente compatible y poder reemplazar al componente base sin interrupciones. Esto garantiza la interoperabilidad entre componentes y evita rupturas en la funcionalidad existente al realizar cambios.

4. Principio de Segregación de la Interfaz

   Este principio enfatiza que las interfaces de los componentes deben estar enfocadas en las necesidades específicas de los usuarios y ser coherentes. En React Native, esto significa que los componentes deben proporcionar interfaces claras y concisas que expongan solo los métodos y propiedades necesarios para su uso. De esta forma, se evita la dependencia innecesaria de funcionalidades no utilizadas y se mejora la modularidad del código.

5. Principio de Inversión de Dependencia

   Este principio establece que los módulos de alto nivel no deben depender directamente de los módulos de bajo nivel, sino de abstracciones. En el caso de React Native, esto implica que las dependencias entre componentes deben basarse en abstracciones en lugar de implementaciones concretas. Esto promueve una arquitectura flexible y desacoplada, lo que facilita la modificación y reutilización de componentes.

Ejemplo del principio Abierto/Cerrado

Se está construyendo una aplicación de comercio electrónico en React Native. Tenemos un componente llamado TarjetaProducto que muestra la información de un producto, incluyendo su nombre, precio y una imagen. Además, se desea agregar la funcionalidad de agregar productos al carrito de compras cuando se hace clic en el componente.

Para aplicar el principio de Abierto/Cerrado, debemos diseñar el componente de manera que sea fácilmente extensible para agregar nuevas funcionalidades sin modificar su código original.

```
import React from 'react';
import { View, Text, Image, TouchableOpacity } from 'react-native';

const TarjetaProducto = ({ nombre, precio, imagen, agregarAlCarrito }) => {
  const alCarrito = () => {
    agregarAlCarrito(nombre, precio);
  };

  return (
    <View>
      <Image source={imagen} />
      <Text>{nombre}</Text>
      <Text>{precio}</Text>
      <TouchableOpacity onPress={alCarrito}>
        <Text>Agregar al Carrito</Text>
      </TouchableOpacity>
    </View>
  );
};

export default TarjetaProducto;
```

En este ejemplo, el componente TarjetaProducto muestra la información básica del producto, como el nombre, el precio y la imagen. Además, proporciona un botón "Agregar al Carrito" que activa la función alCarrito cuando se hace clic.

Ahora, en el escenario que más adelante se desea agregar una nueva funcionalidad al componente para permitir a los usuarios marcar productos como favoritos. En lugar de modificar el código original del componente TarjetaProducto, podemos crear un nuevo componente llamado TarjetaProductoFavorito que extienda el comportamiento del componente original:

```jsx
import React from 'react';
import TarjetaProducto from './TarjetaProducto';

import { Text, TouchableOpacity } from 'react-native';

const TarjetaProductoFavorito = ({ nombre, precio, imagen, agregarAlCarrito, agregarAFavoritos }) => {
  const aFavoritos = () => {
    agregarAFavoritos(nombre);
  };

  return (
    <TarjetaProducto nombre={nombre} precio={precio} imagen={imagen} agregarAlCarrito={agregarAlCarrito}>
      <TouchableOpacity onPress={aFavoritos}>
        <Text>Agregar a Favoritos</Text>
      </TouchableOpacity>
    </TarjetaProducto>
  );
};

export default TarjetaProductoFavorito;
```

En este componente TarjetaProductoFavorito, se utiliza el componente TarjetaProducto original y se agrega la funcionalidad adicional de agregar productos a la lista de favoritos. Al hacer esto, se cumple el principio de Abierto/Cerrado, ya que la funcionalidad se extiende sin modificar el código fuente original del componente TarjetaProducto.

Este enfoque de diseño permite agregar nuevas funcionalidades al componente TarjetaProducto sin afectar su comportamiento original. Además, facilita la reutilización del componente base en diferentes contextos y promueve la modularidad y escalabilidad del código.

Bibliografía

Correa, J. (s.f.). *Aprende Cómo Aplicar los Principios SOLID en React JS*. Obtenido de https://developero.io/blog/react-solid-example

Moniz, E. (12 de 04 de 2021). *Applying SOLID To React*. Obtenido de https://medium.com/docler-engineering/applying-solid-to-react-ca6d1ff926a4

Paez, C. (s.f.). *Aplicando principios SOLID en REACT*. Obtenido de https://github.com/carlos-paezf/SOLID_Principles_React