## ***VISTA***

Es el contenedor principal para organizar otros componentes en la interfaz de usuario. Puede ser utilizado para crear grupos de elementos, definir el estilo de la aplicación y organizar la disposición de otros componentes, similar a un div en desarrollo web. Podemos anidar Vistas para crear estructuras de interfaz de usuario complejas.

Casos de uso comunes:

* Agrupar varios elementos visuales.
* Crear secciones visuales (como encabezados, cuerpos, pies de página).
* Definir estructuras de columnas o filas con flexbox.
* Aplicar estilos como fondo, tamaño, bordes o alineación.

Ejemplo práctico:

*<View>*

*<Texto>¡Hola, React Native!</Texto>*

*</View>*

Explicación:

<View> es el contenedor que agrupa componentes, en este caso, al <Text> que es el componente que se usa para mostrar texto en pantalla. Este código, al ejecutarse en una aplicación de React Native, mostrará simplemente un mensaje en la pantalla:

## ***TEXTO***

Es el componente principal para mostrar texto en nuestra aplicación. Podemos personalizar el texto con diversos estilos, fuentes y opciones de formato.

Casos de uso comunes:

* Mostrar mensajes de bienvenida o instrucciones.
* Crear encabezados o subtítulos.
* Mostrar resultados, errores o notificaciones.
* Mostrar datos dinámicos como nombres, fechas o valores numéricos

Ejemplo práctico:

*< Text style={styles. text }> ¡Bienvenido a mi aplicación!</ Text >*

Explicación:

<Text> es el componente de React Native que se usa para mostrar texto en pantalla. El atributo style={styles.text} le aplica un estilo definido previamente en el objeto styles. "¡Bienvenido a mi aplicación!" es el texto que se va a mostrar. Este código mostrará la frase "¡Bienvenido a mi aplicación!" en la pantalla, con el estilo que esté definido dentro de styles.text.

## ***IMAGEN***

El componente permite cargar y mostrar imágenes desde una URL o desde los recursos de la aplicación. Es útil para mostrar imágenes en la interfaz, con opciones para ajustar el tamaño, ajuste de la imagen y estilo de la imagen, similar a un img en HTML.

Casos de uso:

* Mostrar mensajes.
* Mostrar resultados, errores o notificaciones.

Ejemplo práctico:

*< Image source={ require ( './image.png' )} style={styles. image } />*

Explicación:

Utiliza el componente <Image />. El estilo de la imagen se aplica a través del prop style, que hace referencia a un objeto de estilos (styles.image). require() carga la imagen de manera estática, mientras que los estilos definen propiedades visuales como tamaño y ajuste.

## ***TEXTINPUT***

El componente permite a los usuarios introducir texto. Se utiliza ampliamente para crear campos de entrada, barras de búsqueda y formularios, similar a un input en HTML.

Casos uso:

* Ingresar usuario y contraseña
* Buscar productos o contenido
* Dejar opiniones o retroalimentación

Ejemplo práctico:

*< TextInput*

*placeholder= "Ingrese su nombre de usuario"*

*style={styles.input } onChangeText*

*={ text => onChangeText (texto)}*

*/>*

Explicación:

El componente TextInput permite a los usuarios ingresar texto. En este ejemplo, se utiliza con un placeholder que indica "Ingrese su nombre de usuario", mostrando una sugerencia cuando el campo está vacío. Los estilos se aplican a través de la propiedad style, referenciando un objeto `styles.input` que personaliza la apariencia del campo. La propiedad onChangeText captura y maneja el texto ingresado, pasando el valor a la función onChangeText para actualizar el estado o realizar otras acciones con el texto introducido.

## ***BUTTON***

El componente proporciona una forma sencilla de añadir botones clickeables a nuestra aplicación. Podemos personalizar su apariencia y definir la acción que debe realizar al pulsar el botón, invocando una acción o función determinada en la aplicación, similar a un button en HTML.

Casos de uso:

* Botón para enviar datos en un formulario.
* Botón para cambiar de pantalla o sección.
* Ejecutar funciones como guardar o eliminar datos.

Ejemplo práctico:

*<Button*

*title="Haz clic en mí"*

*onPress={() => alert('¡Botón presionado!')}*

*/>*

Explicación:

Este ejemplo muestra un botón con el texto "Haz clic en mí". Cuando el usuario presiona el botón, se ejecuta una función que muestra una alerta con el mensaje "¡Botón presionado!". Se utiliza la propiedad onPress para manejar el clic y ejecutar la acción.

***SCROLLVIEW***

El componente ScrollView es esencial para crear listas y vistas desplazables. Permite crear áreas de la pantalla que pueden ser desplazadas verticalmente. Es útil para mostrar contenido que no cabe en la pantalla de forma completa, similar a un scrollable container en HTML.

Casos de uso:

* Mostrar listas de elementos que no caben en la pantalla.
* Desplazarse por formularios con muchos campos.
* Mostrar varias imágenes en formato desplazable.

Ejemplo práctico:

*<ScrollView>*

*<Text>¡Desplázate por mí!</Text>*

*{/\* Agrega más contenido aquí \*/}*

*</ScrollView>*

Explicación:

Este ejemplo utiliza el componente ScrollView para crear un área desplazable. Muestra un texto inicial "¡Desplázate por mí!" y permite agregar más contenido dentro de esa área. Si el contenido es más grande que la pantalla, el usuario podrá desplazarlo para verlo completo.

***FLATLIST***

Es una lista vertical con rendimiento optimizado para grandes cantidades de datos. Es ideal para mostrar listas de elementos, con la capacidad de cargar más elementos al desplazarse hacia abajo, similar a un List en HTML.

Casos de uso:

* Mostrar grandes cantidades de elementos sin afectar el rendimiento.
* Cargar más elementos a medida que el usuario se desplaza hacia abajo.
* Mostrar información organizada en filas y columnas.
* Mostrar resultados de búsqueda o listas filtradas.

Ejemplo práctico:

*<FlatList*

*data={data}*

*renderItem={({ item }) => <Text>{item.title}</Text>}*

*keyExtractor={item => item.id.toString()}*

*/>*

Explicación:

El ejemplo muestra cómo usar FlatList para renderizar una lista de elementos. La propiedad data recibe el array de datos, renderItem define cómo mostrar cada elemento (en este caso, mostrando el título del ítem), y keyExtractor asigna una clave única a cada elemento usando su id. Esto permite mostrar listas de manera eficiente.