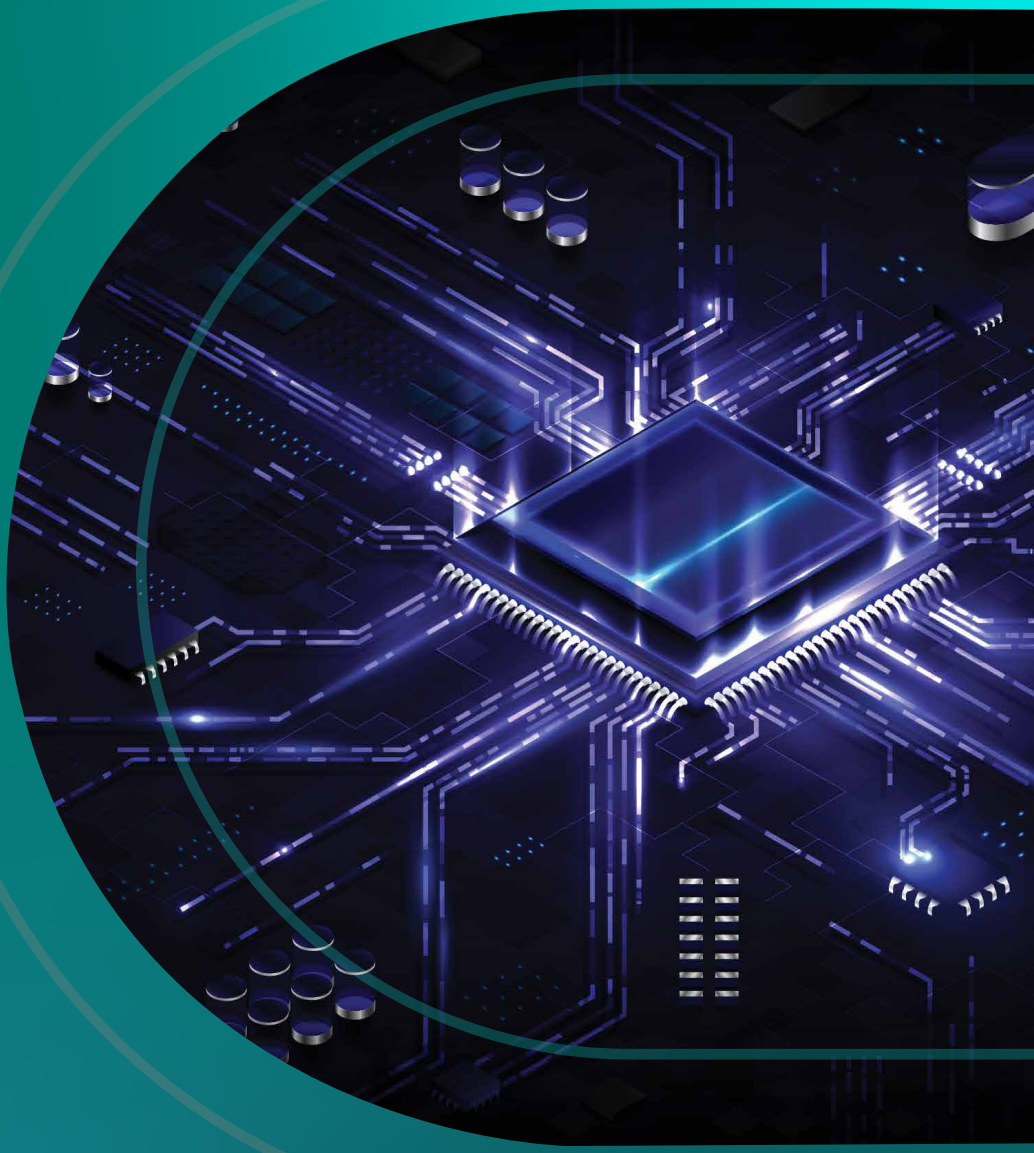
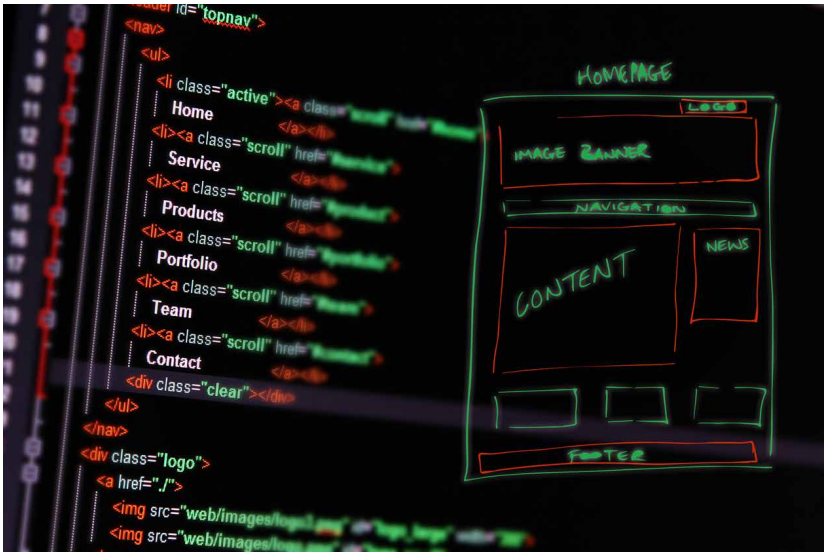


React Nivel Avanzado

Maquetando nuestra App

Introducción



Una interfaz de usuario es el medio, por el cual, un usuario interactúa con una aplicación, un programa o un dispositivo. La interfaz de usuario permite al usuario controlar y manipular el sistema mediante la entrada de información y la visualización de la salida, incluye elementos como: botones, menús, barras de herramientas, campos de texto y gráficos que permiten interactuar con el sistema. También puede incluir elementos como: iconos, ventanas y diálogos que permiten visualizar y manipular la información.

La interfaz de usuario de la aplicación es lo primero con lo que interactúa el usuario, cómo perciba el funcionamiento impacta en lo conocido como experiencia de usuario, por lo que la interfaz debe ser amigable, interactiva y facilitar la accesibilidad de cualquier usuario.

Diseñar la interfaz es una de las etapas más importantes del desarrollo de una aplicación, ya que permite:

- Delimitar lo que la aplicación podrá o no realizar.
- Definir los entregables en cada etapa.
- Asegurar el desarrollo ante cambios en el equipo de desarrollo.
- Mostrar flujos de trabajo de las pantallas.
- Identificar las acciones posibles en cada pantalla.
- Que el proyecto se desarrolle conforme a lo establecido en el diseño.

Una interfaz bien estructurada proporcionará una mejor experiencia a los usuarios.

Debido a esto, se desarrolló un algoritmo llamado Flexbox que posee diferentes opciones, con las cuales, se podrá modificar la alineación de los componentes, justificarlos, cambiar el ancho y alto para que se ajusten de forma automática, dependiendo del dispositivo donde se esté ejecutando, de tal forma, que se juegue con estas opciones hasta que se logre el estilo esperado.

Es importante tomar en cuenta que el diseño es igual de importante que la funcionalidad de la aplicación, para asegurar su éxito.

Explicación

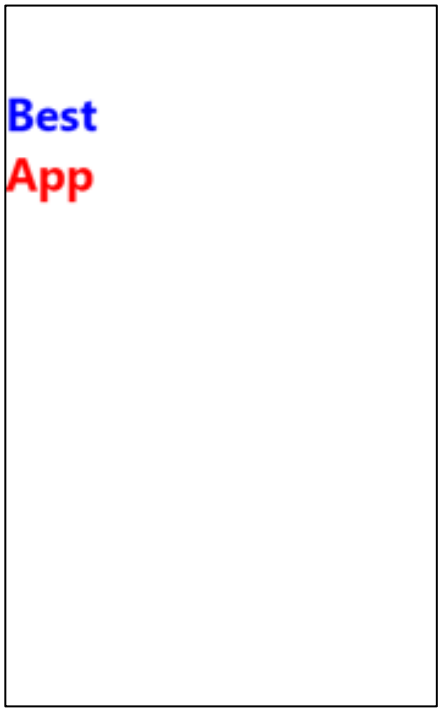
Estilo

De acuerdo con React Native (2023b), el estilo de la aplicación se realizará utilizando JavaScript. Para poder indicar en el código que se quiere dar cierta personalidad o expresión a un componente, se deberá usar la palabra *style*. Si alguna vez se ha programado en Web usando CSS se dará cuenta de que es muy similar.

Imagina que el cliente pide que se utilicen los dos colores que tiene su logo para mostrar el nombre de la aplicación; estos son el azul y el rojo. Para lo solicitado se utilizaría el siguiente código:

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

const LotsOfStyles = () => {
  return (
    <View style={styles.container}>

      <Text style={styles.bigBlue}>Best</Text>
      <Text style={[styles.bigBlue, styles.red]}>App</Text>

    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 50,
  },
  bigBlue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});

export default LotsOfStyles;
```

Explicación



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Usando la propiedad de Style se podrá dar un mejor diseño a la aplicación. Todos los productos siempre son captados por el sentido de la vista, por lo que es igual de importante el diseño como la funcionalidad.

Ancho y alto

Cada dispositivo móvil tiene sus propiedades particulares en cuanto a anchura y altura, así que, al realizar el diseño de la aplicación, se debe considerar esto. Lo recomendable es que se adapte automáticamente a cada dispositivo, esto con el objetivo de que se vea igual de bien en cualquier lado.

Si se realiza una aplicación en la cual el usuario puede comprar boletos de avión; ¿Qué pasaría si los elementos de la aplicación se viesen amontonados?, los usuarios decidirían no utilizar la app, primero, porque no es agradable a la vista y, segundo, no es funcional; deben ser 100% intuitivas y fáciles de usar.

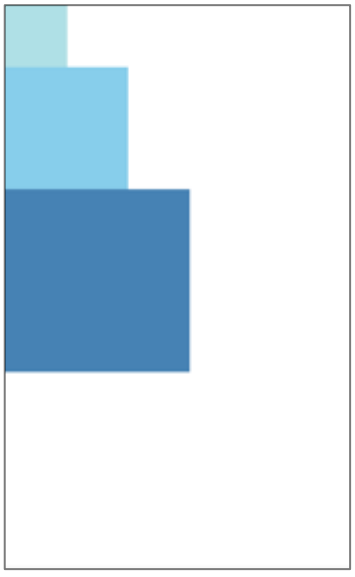
De acuerdo con React Native (2023), para poder cambiar la altura y anchura, se deberá utilizar las propiedades de *width y height* en la de style. Recordando que las pantallas se miden por pixeles.

Explicación

A continuación, se muestra un ejemplo en el que se podrán colocar tres colores en la misma aplicación. Recordar que la altura y la anchura se pueden aplicar en cualquier componente:

```
import React from 'react';
import { View } from 'react-native';

const FixedDimensionsBasics = () => {
  return (
    <View>
      <View style={{
        width: 50, height: 50, backgroundColor: 'powderblue'
      }} />
      <View style={{
        width: 100, height: 100, backgroundColor: 'skyblue'
      }} />
      <View style={{
        width: 150, height: 150, backgroundColor: 'steelblue'
      }} />
    </View>
  );
};

export default FixedDimensionsBasics;
```

El resultado quedaría así:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Layout

Dentro de React Native existe un algoritmo que ayudará a tener el ajuste de los componentes, dependiendo del dispositivo que la app esté utilizando. De acuerdo con React Native (2023a), este algoritmo recibe el nombre de *Flexbox*.

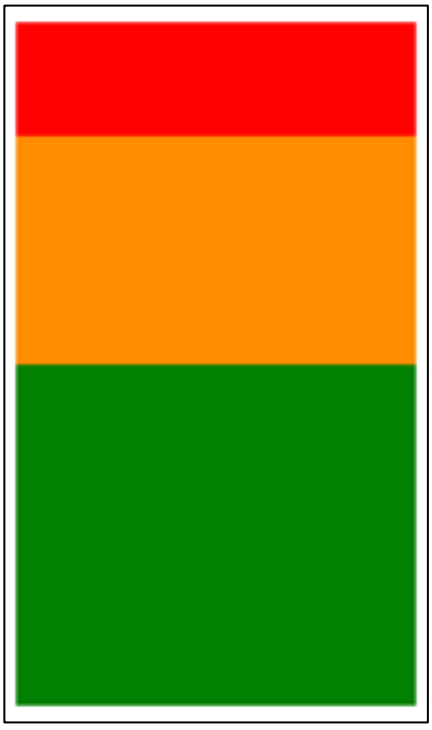
La palabra “flex”, dentro del código, ayudará a definir cómo van a encajar los componentes, conforme al espacio que el dispositivo tenga.

En seguida, se muestra un ejemplo donde cada color utiliza cierto porcentaje de la pantalla. El primer color usa uno, el segundo dos y el tercero tres:

```
import React from "react";
import { StyleSheet, Text, View } from "react-native";

const Flex = () => {
  return (
    <View style={[styles.container, {
      // Try setting `flexDirection` to `"row"`.
      flexDirection: "column"
    }]}>
      <View style={{ flex: 1, backgroundColor: "red" }} />
      <View style={{ flex: 2, backgroundColor: "darkorange" }} />
      <View style={{ flex: 3, backgroundColor: "green" }} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
  },
});

export default Flex;
```

Explicación

Este es el resultado:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Ahora bien, usando flex también se puede definir la dirección en la cual se van a mostrar los componentes; en otras palabras, en qué eje se va a presentar. Para eso, se ha de utilizar la palabra *flexDirection:*

De ese modo, se podrá acomodar de cuatro formas los componentes:

1. **Columna:** Los elementos irán de arriba para bajo.
2. **Fila:** Los elementos irán de derecha a izquierda.
3. **Columna-reversa:** Los elementos irán de abajo para arriba.
4. **Fila-reversa:** Los elementos irán de Izquierda a derecha.

Explicación

Ejemplo de cómo funciona:

```
import React, { useState } from "react";
import { StyleSheet, Text, TouchableOpacity, View } from "react-native";
```

```
const FlexDirectionBasics = () => {
  const [flexDirection, setflexDirection] = useState("column");

  return (
    <PreviewLayout
      label="flexDirection"
      values={["column", "row", "row-reverse", "column-reverse"]}
      selectedValue={flexDirection}
      setSelectedValue={setflexDirection}
    >
      <View
        style={[styles.box, { backgroundColor: "powderblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "skyblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "steelblue" }]}
      />
    </PreviewLayout>
  );
};
```

Explicación

```
const PreviewLayout = ({ label, children, values, selectedValue,
setSelectedValue }) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity
          key={value}
          onPress={() => setSelectedValue(value)}
          style={[
            styles.button,
            selectedValue === value && styles.selected,
          ]}
        >
          <Text
            style={[
              styles.buttonLabel,
              selectedValue === value && styles.selectedLabel,
            ]}
          >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View style={[styles.container, { [label]: selectedValue }]}>
      {children}
    </View>
  </View>
);
```

Explicación

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 8,
    backgroundColor: "aliceblue",
  },
  box: {
    width: 50,
    height: 50,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
  button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default FlexDirectionBasics;
```

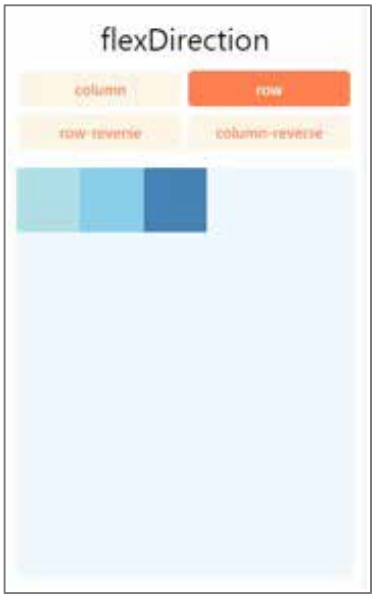
Explicación

Dependiendo del botón que se presione, se obtendrán los siguientes resultados:

1. Columna:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

2. Fila:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

3. Columna- reversa:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

4. Fila-reversa:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Por otro lado, también se puede justificar el contenido dentro de la aplicación. Ejemplo de lo que se puede hacer:

```
import React, { useState } from "react";
import { View, TouchableOpacity, Text, StyleSheet } from
"react-native";

const JustifyContentBasics = () => {
  const [justifyContent, setJustifyContent] =
useState("flex-start");

  return (
    <PreviewLayout
      label="justifyContent"
      selectedValue={justifyContent}
      values={[
        "flex-start",
        "flex-end",
        "center",
        "space-between",
        "space-around",
        "space-evenly",
      ]}
      setSelectedValue={setJustifyContent}
    >
      <View style={[styles.box, { backgroundColor: "powderblue" }]}
/>
      <View style={[styles.box, { backgroundColor: "skyblue" }]} />
      <View style={[styles.box, { backgroundColor: "steelblue" }]}
/>
    </PreviewLayout>
  );
};
```

Explicación

```
const PreviewLayout = ({
  label,
  children,
  values,
  selectedValue,
  setSelectedValue,
}) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity
          key={value}
          onPress={() => setSelectedValue(value)}
          style={[styles.button, selectedValue === value &&
styles.selected]}
        >
          <Text
            style={[
              styles.buttonLabel,
              selectedValue === value && styles.selectedLabel,
            ]}
          >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View style={[styles.container, { [label]: selectedValue }]}>
      {children}
    </View>
  </View>
);
```

Explicación

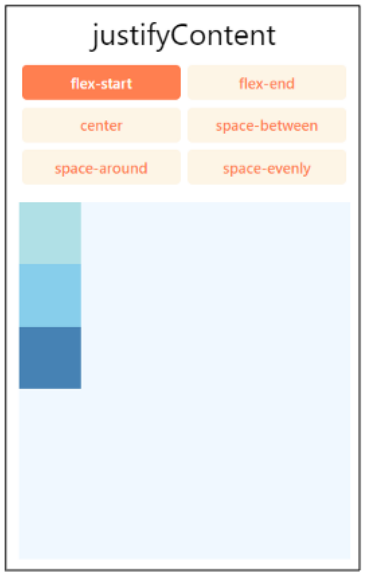
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 8,
    backgroundColor: "aliceblue",
  },
  box: {
    width: 50,
    height: 50,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
```

Explicación

```
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default JustifyContentBasics;
```

Estos son los resultados, de acuerdo, con el botón que se presione:

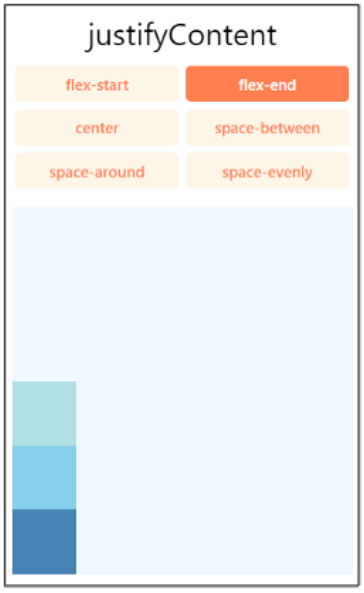
1. Empezar-Flex:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

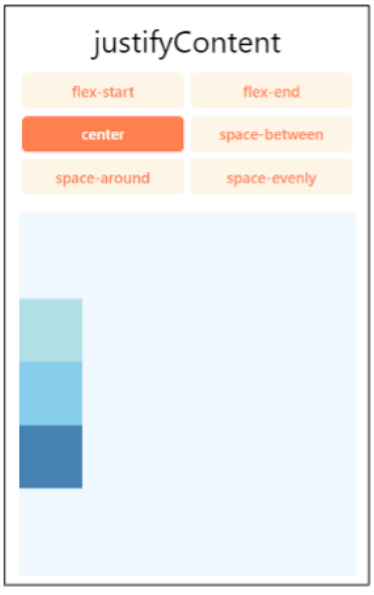
Explicación

2. Terminar-Flex:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

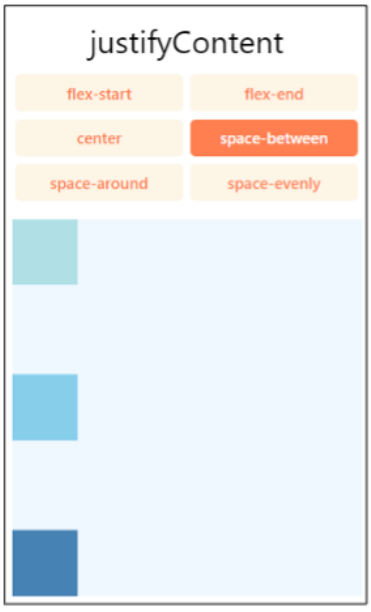
3. Centrado:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

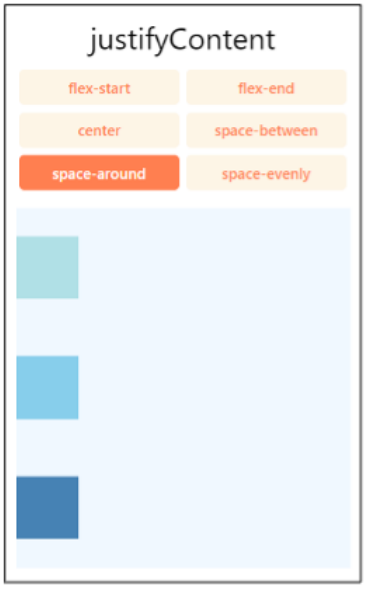
Explicación

4. Espacio-entre:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

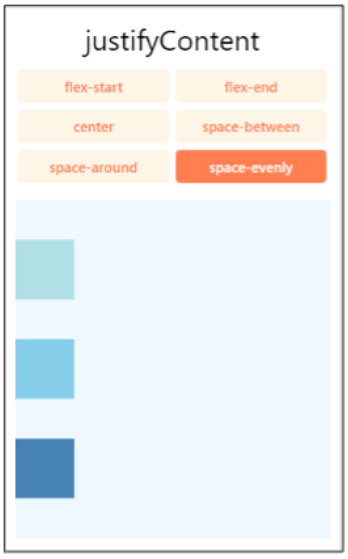
5. Espacio-alrededor:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

6. Espacio- uniforme:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Otro aspecto importante es que se puede alinear el contenido de la aplicación, esto se puede lograr con *alignContent*; se hará centrándose en el eje transversal. Enseguida se encontrará un ejemplo de su funcionamiento:

```
import React, { useState } from "react";
import { View, TouchableOpacity, Text, StyleSheet } from
"react-native";

const AlignContentLayout = () => {
  const [alignContent, setAlignContent] = useState("flex-start");

  return (
    <PreviewLayout
      label="alignContent"
      selectedValue={alignContent}
      values={[
        "flex-start",
        "flex-end",
        "stretch",
        "center",
        "space-between",
        "space-around",
      ]}
```

Explicación

```
      setSelectedValue={setAlignContent}>
      <View style={[styles.box, { backgroundColor: "orangered" }]}
/>
      <View style={[styles.box, { backgroundColor: "orange" }]} />
      <View style={[styles.box, { backgroundColor: "mediumseagreen"
}]} />
      <View style={[styles.box, { backgroundColor: "deepskyblue" }]}
/>
      <View style={[styles.box, { backgroundColor: "mediumturquoise"
}]} />
      <View style={[styles.box, { backgroundColor: "mediumslateblue"
}]} />
      <View style={[styles.box, { backgroundColor: "purple" }]} />
    </PreviewLayout>
  );
};
```

```
const PreviewLayout = ({
  label,
  children,
  values,
  selectedValue,
  setSelectedValue,
}) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity
          key={value}
          onPress={() => setSelectedValue(value)}
          style={[
            styles.button,
            selectedValue === value && styles.selected,
          ]}
        >
```

Explicación

```
            <Text style={[
                styles.buttonLabel,
                selectedValue === value &&
                styles.selectedLabel,
              ]} >
              {value}
            </Text>
          </TouchableOpacity>
        ))}
      </View>
      <View
        style={[
          styles.container,
          { [label]: selectedValue },
        ]}
      >
        {children}
      </View>
    </View>
  );
```

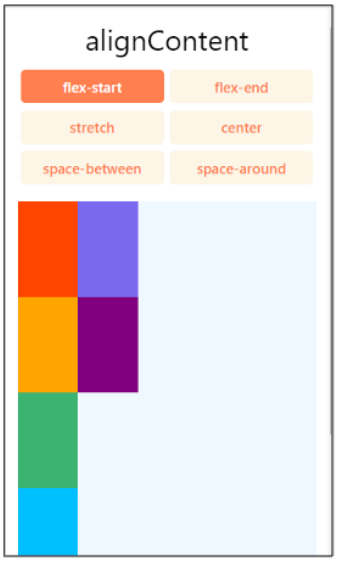
Explicación

```
  button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default AlignContentLayout;
```

Explicación

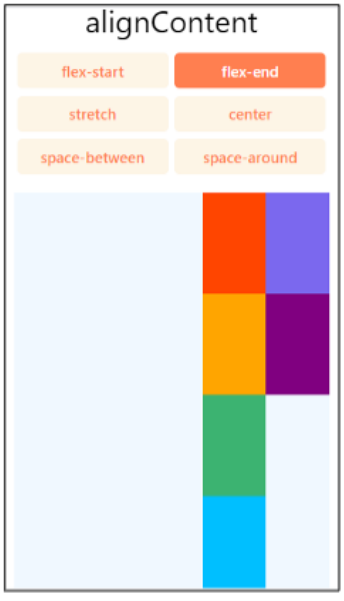
Los resultados se obtienen de acuerdo con el botón que se presione:

1. Empezar-Flex:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

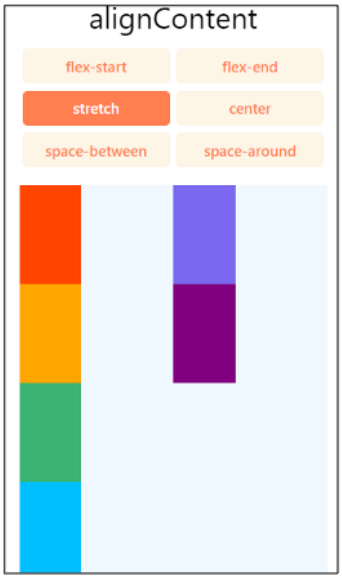
2. Terminar-Flex:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

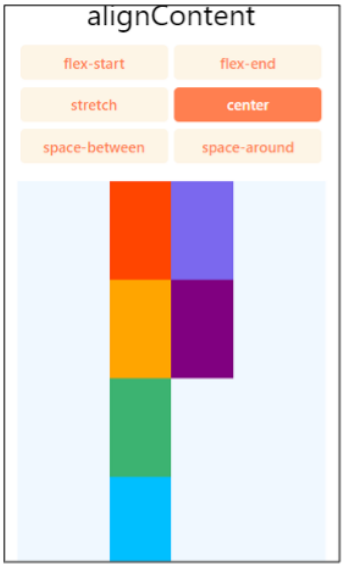
Explicación

3. Estrecho:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

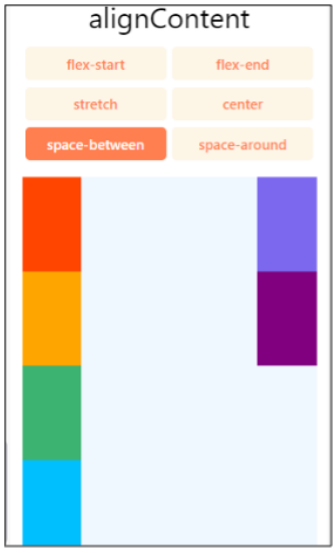
4. Centrado:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

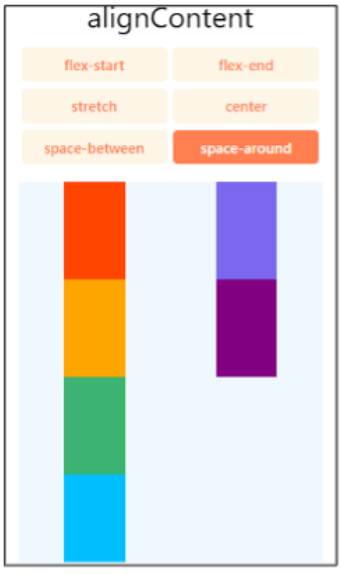
Explicación

5. Espacio-entre:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

6. Espacio-alrededor:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Por otro lado, también existe la propiedad de *flexWrap*, que ajusta los componentes que superan el tamaño del eje principal. Ahora, se muestra un ejemplo de cómo funciona:

```
import React, { useState } from "react";
import { View, TouchableOpacity, Text, StyleSheet } from "react-native";

const FlexWrapLayout = () => {
  const [flexWrap, setFlexWrap] = useState("wrap");

  return (
    <PreviewLayout
      label="flexWrap"
      selectedValue={flexWrap}
      values={["wrap", "nowrap"]}
      setSelectedValue={setFlexWrap}>
      <View style={[styles.box, { backgroundColor: "orangered" }]} />
      <View style={[styles.box, { backgroundColor: "orange" }]} />
      <View style={[styles.box, { backgroundColor: "mediumseagreen" }]}
/>
      <View style={[styles.box, { backgroundColor: "deepskyblue" }]} />
      <View style={[styles.box, { backgroundColor: "mediumturquoise" }]}
/>
      <View style={[styles.box, { backgroundColor: "mediumslateblue" }]}
/>
      <View style={[styles.box, { backgroundColor: "purple" }]} />
    </PreviewLayout>
  );
};
```

```
  const styles = StyleSheet.create({
    container: {
      flex: 1,
      flexWrap: "wrap",
      marginTop: 8,
      backgroundColor: "aliceblue",
      maxHeight: 400,
    },
    box: {
      width: 50,
      height: 80,
    },
    row: {
      flexDirection: "row",
      flexWrap: "wrap",
    },
```

Explicación

```
const PreviewLayout = ({ label, children, values, selectedValue,
setSelectedValue }) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity key={value}
          onPress={() => setSelectedValue(value)}
          style={[ styles.button, selectedValue === value &&
styles.selected ]}
        >
          <Text style={[ styles.buttonLabel, selectedValue === value &&
styles.selectedLabel ]}>
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View style={[ styles.container, { [label]: selectedValue }]}>
      {children}
    </View>
  </View>
);
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 8,
    backgroundColor: "aliceblue",
    maxHeight: 400,
  },
  box: {
    width: 50,
    height: 80,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
```

Explicación

```
button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default FlexWrapLayout;
```

Explicación

De acuerdo con el botón que se presione, se obtienen los siguientes resultados:

1. Wrap:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

2. No-wRAP:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Por último, se revisará cómo manejar automáticamente la altura y anchura de los componentes para adaptarlos a cada pantalla. Ejemplo:

```
import React, { useState } from "react";
import {
  View,
  SafeAreaView,
  TouchableOpacity,
  Text,
  StyleSheet,
} from "react-native";

const WidthHeightBasics = () => {
  const [widthType, setWidthType] = useState("auto");
  const [heightType, setHeightType] = useState("auto");

  return (
    <PreviewLayout
      widthType={widthType}
      heightType={heightType}
      widthValues={["auto", 300, "80%"]}
      heightValues={["auto", 200, "60%"]}
      setWidthType={setWidthType}
      setHeightType={setHeightType}
    >
       <View style={{
          alignSelf: "flex-start",
          backgroundColor: "aliceblue",
          height: heightType,
          width: widthType,
          padding: 15,
        }}
       >
```

Explicación

```
        <View style={[ styles.box, { backgroundColor: "powderblue" } ]}
/>
        <View style={[ styles.box, { backgroundColor: "skyblue" } ]} />
        <View style={[ styles.box, { backgroundColor: "steelblue" } ]} />
      </View>
    </PreviewLayout>
  );
};

const PreviewLayout = ({ children, widthType, heightType, widthValues,
heightValues,
setWidthType, setHeightType }) => (
  <SafeAreaView style={{ flex: 1, padding: 10 }}>
    <View style={styles.row}>
      <Text style={styles.label}>width </Text>
      {widthValues.map((value) => (
        <TouchableOpacity key={value}
          onPress={() => setWidthType(value)}
          style={[ styles.button, widthType === value && styles.selected
]}
        >
          <Text style={[ styles.buttonLabel, widthType === value &&
styles.selectedLabel ]} >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View style={styles.row}>
      <Text style={styles.label}>height </Text>
      {heightValues.map((value) => (
        <TouchableOpacity key={value}
          onPress={() => setHeightType(value)}
          style={[ styles.button, heightType === value && styles.selected
]}
        >
          <Text style={[ styles.buttonLabel, heightType === value &&
styles.selectedLabel ]} >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    {children}
  </SafeAreaView>
);
```

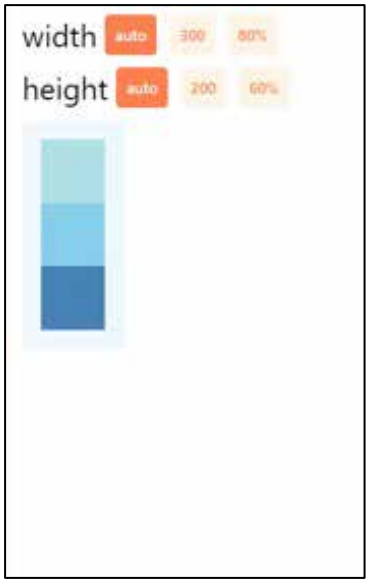
Explicación

```
const styles = StyleSheet.create({
  box: {
    width: 50,
    height: 50,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
  button: {
    padding: 8,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginRight: 10,
    marginBottom: 10,
  },
  selected: {
    backgroundColor: "coral",
    shadowOpacity: 0,
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
```

```
label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default WidthHeightBasics;
```

Explicación

El resultado sería el siguiente:



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

El ciclo de vida del desarrollo de software (SDLC) describe varias tareas necesarias, para crear una aplicación de software, entre ellas, el diseño de la aplicación.

De acuerdo con AWS (s.f.), en esta etapa, los ingenieros de software analizan los requisitos e identifican las mejores soluciones para crear el software. Por ejemplo, pueden plantearse la integración de módulos ya existentes, elegir la tecnología e identificar herramientas de desarrollo. Decidirán la mejor manera de integrar el nuevo software en cualquier infraestructura de TI existente que la organización pueda tener.

Cierre

Dentro del desarrollo de cualquier producto de software, es importante poner atención en su diseño, ya que, además de ser tan importante como la funcionalidad, el diseño influye en la aceptación que tendrá por parte del usuario final y esto, en conjunto con la funcionalidad, deben proporcionar una experiencia de usuario agradable.

El uso de Flexbox permite mejorar el comportamiento de la aplicación para adaptarla a las diferentes resoluciones de pantallas en que se pueda presentar, sin afectar de forma importante el diseño y el orden en que se colocan los elementos en la interfaz.

Referencias Bibliográficas

- AWS. (s.f.). *¿Que es SDLC?* Recuperado de https://aws.amazon.com/es/what-is/sdlc/
- React Native. (2023). *Height and Width.* Recuperado de https://reactnative.dev/docs/height-and-width
- React Native. (2023a). *Layout with Flexbox.* Recuperado de https://reactnative.dev/docs/flexbox
- React Native. (2023b). *Style.* Recuperado de https://reactnative.dev/docs/style

Para saber más

Lecturas

Para conocer más acerca de **Diseño de la aplicación**, te sugerimos leer lo siguiente:

- React Native. (2023). *Color Reference.* Recuperado de https://reactnative.dev/docs/colors
- React Native. (2023). *Images.* Recuperado de https://reactnative.dev/docs/images

Videos

Para conocer más acerca de **Diseño de la aplicación** te recomendamos:

- BetoMoedano. (2022, 12 de enero). *Flexbox* | React native Tutorial [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=1MDcMdTTuxg
- BetoMoedano. (2022, 5 de junio). *React Native Expo | Tutorial para principiantes 2022* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=QoKr_RJgd3E

Checkpoint

Asegúrate de:

- Comprender las ventajas de Flexbox para el desarrollo de software.
- Reconocer la sintaxis de las opciones con las que cuenta Flexbox, para el diseño de aplicaciones.
- Entender el uso de cada opción de Flexbox para el diseño de aplicaciones.

Requerimientos técnicos

- Android Studio.
- React.
- Xcode.
- VS Code.
- Figma (ver Prework Tema 4).

Prework

- Deberás revisar los siguientes temas:
  - **Tema 1. Fundamentos de React.**
  - **Tema 2. Primeros pasos con React Native.**
  - **Tema 3. Componentes Core y Nativos.**