Aplikacje mobilne

ReactNative - design

Mateusz Pawełkiewicz

1) Stylowanie

W React Native stylizujesz swoją aplikację przy użyciu języka JavaScript. Wszystkie główne komponenty przyjmują właściwość o nazwie "style". Nazwy i wartości stylów zazwyczaj odpowiadają temu, jak działa CSS w środowisku internetowym, z wyjątkiem tego, że nazwy są zapisywane w notacji camel case, na przykład "backgroundColor" zamiast "background-color".

Właściwość "style" może być zwykłym obiektem JavaScript. To jest to, co zazwyczaj używamy w kodzie przykładowym. Możesz także przekazać tablicę stylów - ostatni styl w tablicy ma pierwszeństwo, dzięki czemu możesz dziedziczyć style.

W miarę jak komponent staje się bardziej złożony, często jest bardziej czytelnie używać metody "StyleSheet.create", aby zdefiniować wiele stylów w jednym miejscu. Oto przykład:

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

const LotsOfStyles = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.red}>just red</Text>
      <Text style={styles.bigBlue}>just bigBlue</Text>
      <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>
      <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>
    </View>
  );
};

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

Jednym z często spotykanych wzorców jest umożliwienie komponentowi przyjmowania właściwości "style", które z kolei są używane do stylizacji podkomponentów. Możesz użyć tego mechanizmu, aby style "kaskadowały" tak, jak dzieje się to w CSS.

Istnieje wiele innych sposobów dostosowywania stylu tekstu. Sprawdź pełną listę w referencji komponentu "Text".

Teraz możesz uczynić swój tekst piękniejszym. Kolejnym krokiem w staniu się ekspertem od stylizacji jest nauka kontroli rozmiaru komponentu.

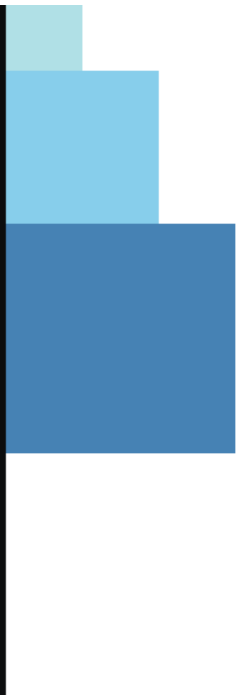
2) Wysokość i Szerokość

a) Stałe wymiary

Wysokość i szerokość komponentu określają jego rozmiar na ekranie.

Ogólnym sposobem na ustawienie wymiarów komponentu jest dodanie stałej szerokości i wysokości do właściwości "style". Wszystkie wymiary w React Native są bezjednostkowe i reprezentują niezależne od gęstości piksele.

```
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',
        }}
```



Ustawianie wymiarów w ten sposób jest powszechne dla komponentów, których rozmiar powinien zawsze być stały i nie obliczać się na podstawie rozmiaru ekranu.

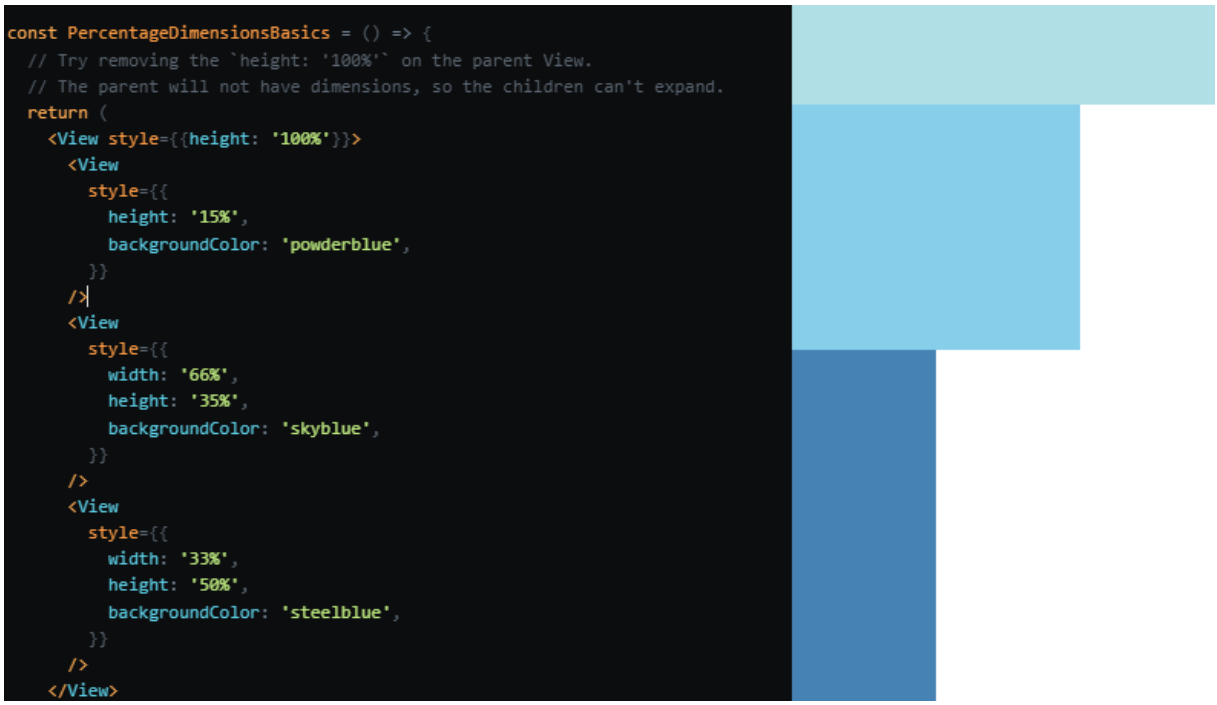
b) Flex

Flex w stylizacji komponentu używamy, aby komponent automatycznie rozszerzał się i kurczył w zależności od dostępnego miejsca. Zazwyczaj używa się flex: 1, co oznacza, że komponent wypełni całą dostępną przestrzeń, równo dzieląc ją między innymi komponentami w tym samym rodzicu. Im większa wartość flex, tym większą część przestrzeni zajmie komponent w porównaniu do swoich rodzeństwa.

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

const FlexDimensionsBasics = () => {
  return (
    // Try removing the `flex: 1` on the parent View.
    // The parent will not have dimensions, so the children can't expand.
    // What if you add `height: 300` instead of `flex: 1`?
    <View style={{flex: 1}}>
      <View style={{flex: 1, backgroundColor: 'powderblue'}} />
      <View style={{flex: 2, backgroundColor: 'skyblue'}} />
      <View style={{flex: 3, backgroundColor: 'steelblue'}} />
    </View>
  );
};

export default FlexDimensionsBasics;
```

c) Procentowe

Jeśli chcesz wypełnić określoną część ekranu, ale nie chcesz używać elastycznego układu (flex), możesz użyć wartości procentowych w stylizacji komponentu. Podobnie jak w przypadku wymiarów elastycznych, wymiary procentowe wymagają rodzica o określonym rozmiarze.



```
const PercentageDimensionsBasics = () => {
  // Try removing the `height: '100%'` on the parent View.
  // The parent will not have dimensions, so the children can't expand.
  return (
    <View style={{height: '100%'}}>
      <View
        style={{
          height: '15%',
          backgroundColor: 'powderblue',
        }}
      />
      <View
        style={{
          width: '66%',
          height: '35%',
          backgroundColor: 'skyblue',
        }}
      />
      <View
        style={{
          width: '33%',
          height: '50%',
          backgroundColor: 'steelblue',
        }}
      />
    </View>
```

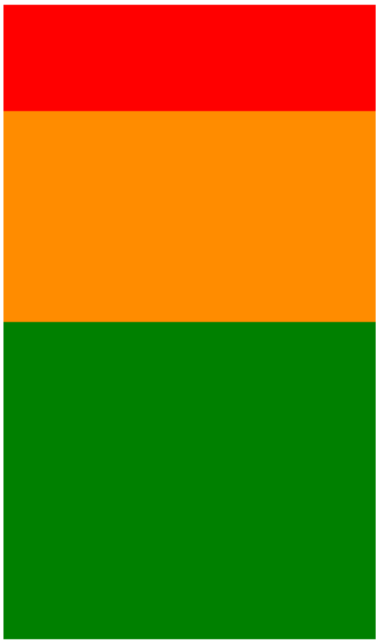
3) Flexbox

a) Flex

Właściwość "flex" definiuje, w jaki sposób elementy zostaną "wypełnione" dostępną przestrzenią wzdłuż głównej osi. Przestrzeń będzie podzielona zgodnie z właściwością "flex" każdego elementu.

W poniższym przykładzie widoki czerwony, pomarańczowy i zielony są wszystkimi dziećmi w widoku kontenera, który ma ustawioną właściwość "flex: 1". Widok czerwony używa "flex: 1", widok pomarańczowy używa "flex: 2", a widok zielony używa "flex: 3". 1+2+3 = 6, co oznacza, że widok czerwony otrzyma 1/6 przestrzeni, widok pomarańczowy 2/6 przestrzeni, a widok zielony 3/6 przestrzeni.

```jsx
import React from 'react';
import {StyleSheet, 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,
  },
});
```



b) Flex Direction

Właściwość "flexDirection" kontroluje kierunek, w którym układane są dzieci węzła. Jest to także nazywane główną osią. Oś poprzeczna jest osią prostopadłą do głównej osi lub osią, wzdłuż której układane są linie zawijania.

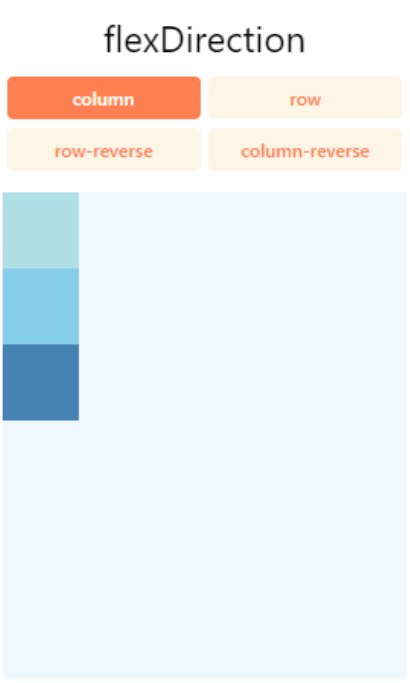
- "column" (wartość domyślna): Dzieci są układane od góry do dołu. Jeśli włączona jest zawijanie, to kolejna linia zacznie się po prawej stronie pierwszego elementu na górze kontenera.

- "row": Dzieci są układane od lewej do prawej. Jeśli włączone jest zawijanie, to kolejna linia zacznie się pod pierwszym elementem po lewej stronie kontenera.

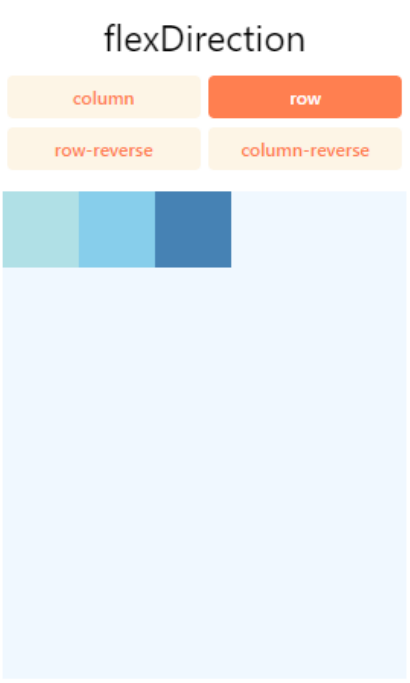
- "column-reverse": Dzieci są układane od dołu do góry. Jeśli włączona jest opcja zawijania, to kolejna linia zacznie się po prawej stronie pierwszego elementu na dole kontenera.

- "row-reverse": Dzieci są układane od prawej do lewej. Jeśli włączone jest zawijanie, to kolejna linia zacznie się pod pierwszym elementem po prawej stronie kontenera.

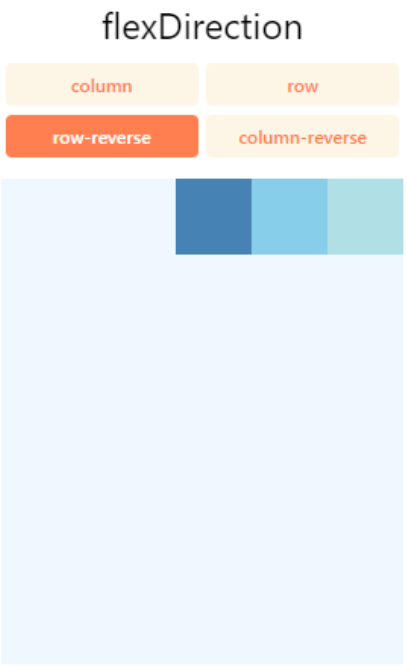
```tsx
import React, {useState} from 'react';
import {StyleSheet, Text, TouchableOpacity, View} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};

type PreviewLayoutProps = PropsWithChildren<{
  label: string;
  values: string[];
  selectedValue: string;
  setSelectedValue: (value: string) => void;
}>;
```



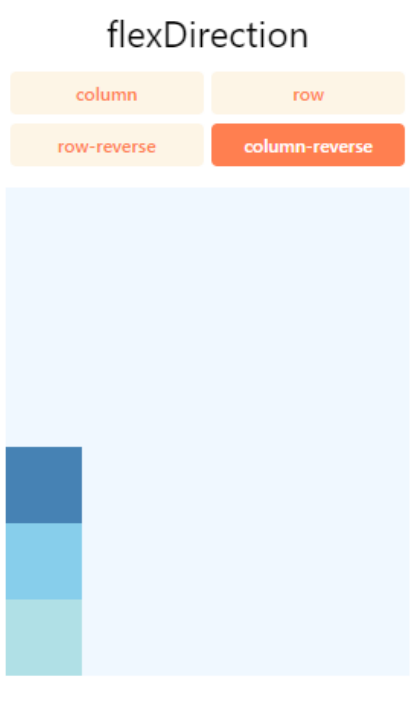

```tsx
import React, {useState} from 'react';
import {StyleSheet, Text, TouchableOpacity, View} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};

type PreviewLayoutProps = PropsWithChildren<{
  label: string;
  values: string[];
  selectedValue: string;
  setSelectedValue: (value: string) => void;
}>;
```



```tsx
import React, {useState} from 'react';
import {StyleSheet, Text, TouchableOpacity, View} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};

type PreviewLayoutProps = PropsWithChildren<{
  label: string;
  values: string[];
  selectedValue: string;
  setSelectedValue: (value: string) => void;
}>;
```




```tsx
import React, {useState} from 'react';
import {StyleSheet, Text, TouchableOpacity, View} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};

type PreviewLayoutProps = PropsWithChildren<{
  label: string;
  values: string[];
  selectedValue: string;
  setSelectedValue: (value: string) => void;
}>;
```

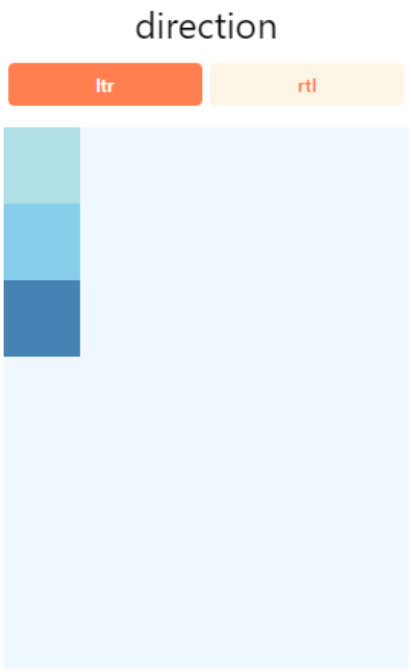
c) Layout Direction

Kierunek układu określa, w jakim kierunku powinny być układane dzieci i tekst w hierarchii. Kierunek układu wpływa również na to, co oznaczają krawędzie "start" i "end". Domyślnie React Native układa elementy w kierunku od lewej do prawej (LTR). W tym trybie "start" odnosi się do lewej strony, a "end" odnosi się do prawej strony.

- LTR (wartość domyślna): Tekst i dzieci są układane od lewej do prawej. Marginesy i wcięcia stosowane do początku elementu są stosowane po lewej stronie.

- RTL: Tekst i dzieci są układane od prawej do lewej. Marginesy i wcięcia stosowane do początku elementu są stosowane po prawej stronie.

```tsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

const DirectionLayout = () => {
  const [direction, setDirection] = useState('ltr');

  return (
    <PreviewLayout
      label="direction"
      selectedValue={direction}
      values={['ltr', 'rtl']}
      setSelectedValue={setDirection}>
      <View style={[styles.box, {backgroundColor: 'powderblue'}]} />
      <View style={[styles.box, {backgroundColor: 'skyblue'}]} />
      <View style={[styles.box, {backgroundColor: 'steelblue'}]} />
    </PreviewLayout>
  );
};

type PreviewLayoutProps = PropsWithChildren<{
  label: string;
  values: string[];
  selectedValue: string;
  setSelectedValue: (value: string) => void;
}>;
```




```tsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

const DirectionLayout = () => {
  const [direction, setDirection] = useState('ltr');

  return (
    <PreviewLayout
      label="direction"
      selectedValue={direction}
      values={['ltr', 'rtl']}
      setSelectedValue={setDirection}>
      <View style={[styles.box, {backgroundColor: 'powderblue'}]} />
      <View style={[styles.box, {backgroundColor: 'skyblue'}]} />
      <View style={[styles.box, {backgroundColor: 'steelblue'}]} />
    </PreviewLayout>
  );
};

type PreviewLayoutProps = PropsWithChildren<{
  label: string;
  values: string[];
  selectedValue: string;
  setSelectedValue: (value: string) => void;
}>;
```

d) Justify Content

Właściwość "justifyContent" opisuje, jak wyrównać dzieci wzdłuż głównej osi ich kontenera. Możesz użyć tej właściwości na przykład do wyśrodkowania dziecka w poziomie w kontenerze, w którym ustawiono właściwość "flexDirection" na "row", lub w pionie w kontenerze z właściwością "flexDirection" ustawioną na "column".

- "flex-start" (wartość domyślna): Wyrównuje dzieci kontenera do początku głównej osi kontenera.

- "flex-end": Wyrównuje dzieci kontenera do końca głównej osi kontenera.

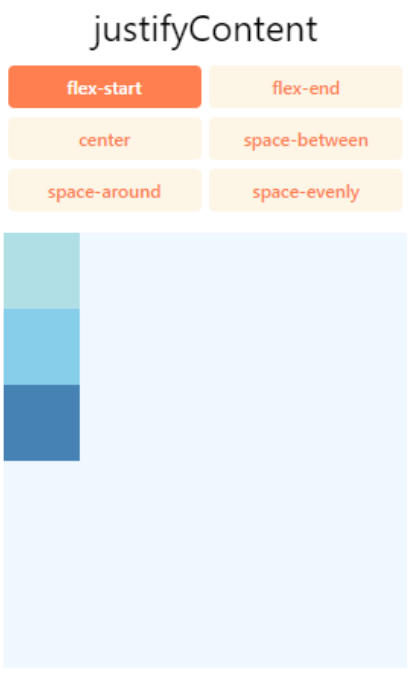
- "center": Wyśrodkowuje dzieci kontenera wzdłuż głównej osi kontenera.

- "space-between": Rozmieszcza dzieci równomiernie wzdłuż głównej osi kontenera, dzieląc pozostałą przestrzeń między dziećmi.

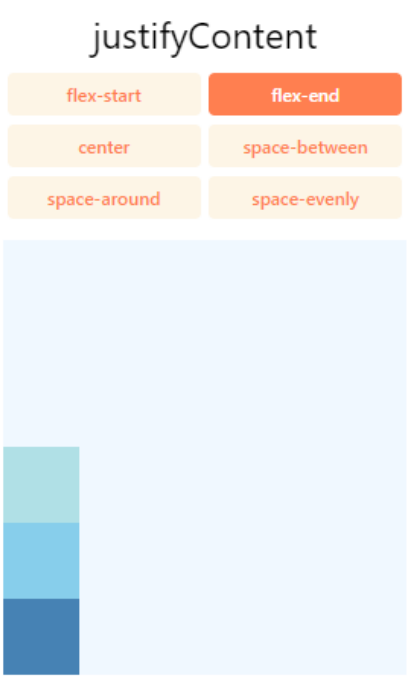
- "space-around": Rozmieszcza dzieci równomiernie wzdłuż głównej osi kontenera, dzieląc pozostałą przestrzeń wokół dzieci. W porównaniu do "space-between", "space-around" spowoduje, że przestrzeń zostanie rozdzielona na początku pierwszego dziecka i na końcu ostatniego dziecka.

- "space-evenly": Rozmieszcza dzieci równomiernie w kontenerze wzdłuż głównej osi. Odległość między każdą parą sąsiadujących elementów, krawędzią główną początkową a pierwszym elementem oraz krawędzią główną końcową a ostatnim elementem, jest identyczna.

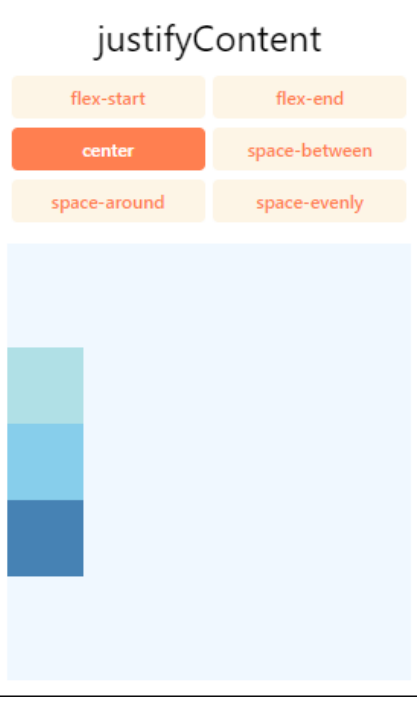
```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};
```



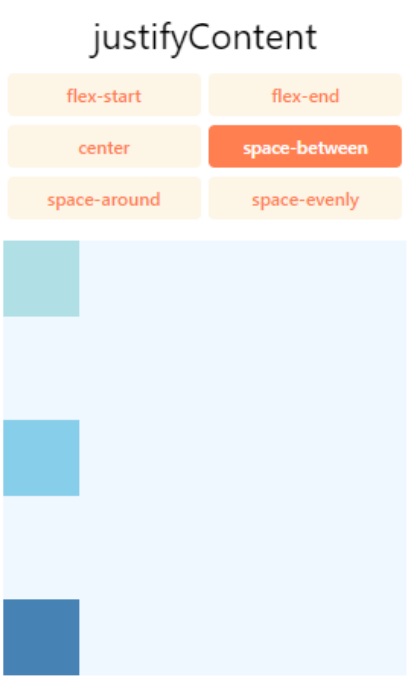
```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};
```



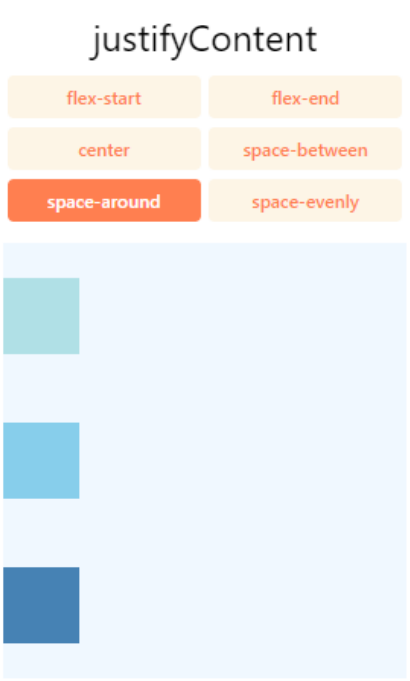

```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};
```



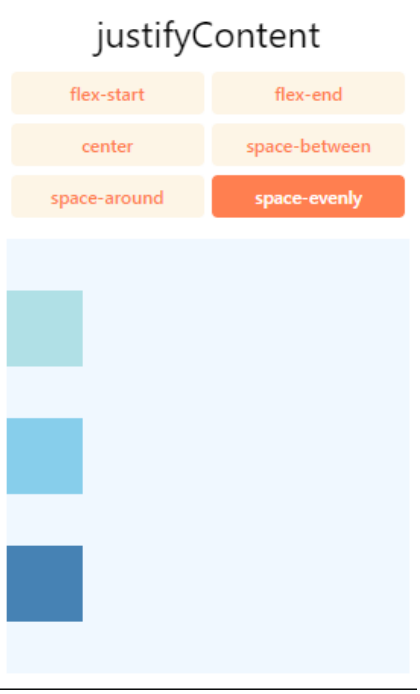
```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};
```




```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};
```



```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

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>
  );
};
```



e) Align Items

Właściwość "alignItems" opisuje, w jaki sposób wyrównać dzieci względem osi poprzecznej (cross axis) ich kontenera. Jest bardzo podobna do właściwości "justifyContent", ale zamiast dotyczyć głównej osi, "alignItems" dotyczy osi poprzecznej.

- "stretch" (wartość domyślna): Rozciąga dzieci kontenera, aby dopasować się do wysokości osi poprzecznej kontenera.

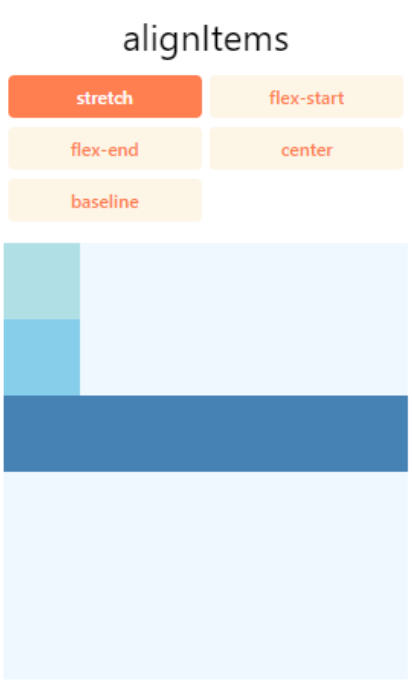
- "flex-start": Wyrównuje dzieci kontenera do początku osi poprzecznej kontenera.

- "flex-end": Wyrównuje dzieci kontenera do końca osi poprzecznej kontenera.

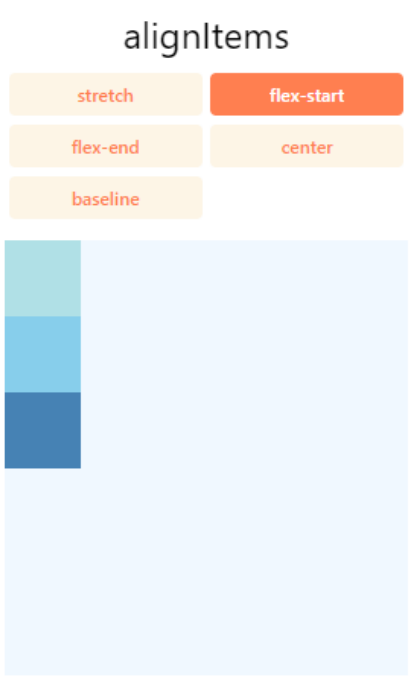
- "center": Wyśrodkowuje dzieci kontenera względem osi poprzecznej kontenera.

- "baseline": Wyrównuje dzieci kontenera względem wspólnego linii bazowej. Indywidualne dzieci mogą być ustawione jako punkty odniesienia dla linii bazowej dla swoich rodziców.

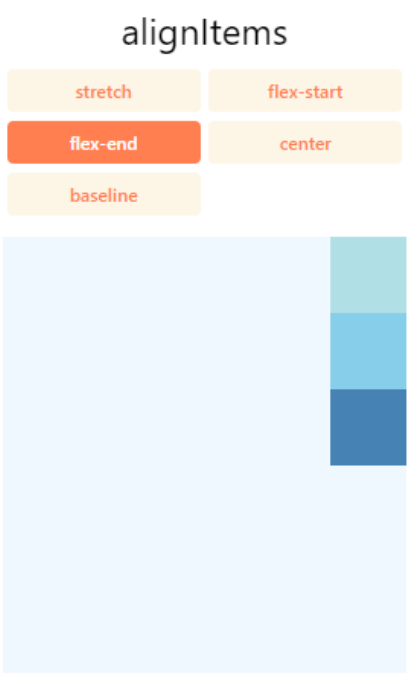
```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

const AlignItemsLayout = () => {
  const [alignItems, setAlignItems] = useState('stretch');

  return (
    <PreviewLayout
      label="alignItems"
      selectedValue={alignItems}
      values={['stretch', 'flex-start', 'flex-end', 'center', 'baseline']}
      setSelectedValue={setAlignItems}>
      <View style={[styles.box, {backgroundColor: 'powderblue'}]} />
      <View style={[styles.box, {backgroundColor: 'skyblue'}]} />
      <View
        style={[
          styles.box,
          {
            backgroundColor: 'steelblue',
            width: 'auto',
            minWidth: 50,
          },
        ]}
      />
    </PreviewLayout>
```



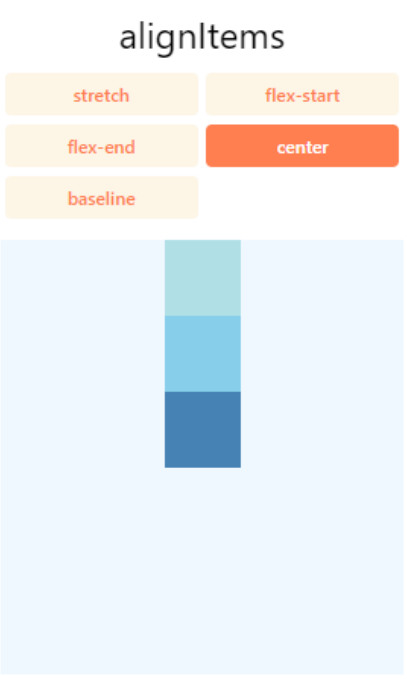

```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

const AlignItemsLayout = () => {
  const [alignItems, setAlignItems] = useState('stretch');

  return (
    <PreviewLayout
      label="alignItems"
      selectedValue={alignItems}
      values={['stretch', 'flex-start', 'flex-end', 'center', 'baseline']}
      setSelectedValue={setAlignItems}>
      <View style={[styles.box, {backgroundColor: 'powderblue'}]} />
      <View style={[styles.box, {backgroundColor: 'skyblue'}]} />
      <View
        style={[
          styles.box,
          {
            backgroundColor: 'steelblue',
            width: 'auto',
            minWidth: 50,
          },
        ]}
      />
    </PreviewLayout>
```



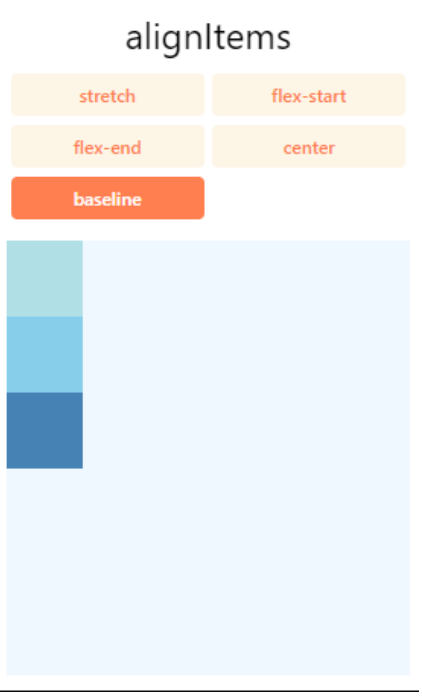
```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

const AlignItemsLayout = () => {
  const [alignItems, setAlignItems] = useState('stretch');

  return (
    <PreviewLayout
      label="alignItems"
      selectedValue={alignItems}
      values={['stretch', 'flex-start', 'flex-end', 'center', 'baseline']}
      setSelectedValue={setAlignItems}>
      <View style={[styles.box, {backgroundColor: 'powderblue'}]} />
      <View style={[styles.box, {backgroundColor: 'skyblue'}]} />
      <View
        style={[
          styles.box,
          {
            backgroundColor: 'steelblue',
            width: 'auto',
            minWidth: 50,
          },
        ]}
      />
    </PreviewLayout>
```




```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

const AlignItemsLayout = () => {
  const [alignItems, setAlignItems] = useState('stretch');

  return (
    <PreviewLayout
      label="alignItems"
      selectedValue={alignItems}
      values={['stretch', 'flex-start', 'flex-end', 'center', 'baseline']}
      setSelectedValue={setAlignItems}>
      <View style={[styles.box, {backgroundColor: 'powderblue'}]} />
      <View style={[styles.box, {backgroundColor: 'skyblue'}]} />
      <View
        style={[
          styles.box,
          {
            backgroundColor: 'steelblue',
            width: 'auto',
            minWidth: 50,
          },
        ]}
      />
    </PreviewLayout>
```



```jsx
import React, {useState} from 'react';
import {View, TouchableOpacity, Text, StyleSheet} from 'react-native';
import type {PropsWithChildren} from 'react';

const AlignItemsLayout = () => {
  const [alignItems, setAlignItems] = useState('stretch');

  return (
    <PreviewLayout
      label="alignItems"
      selectedValue={alignItems}
      values={['stretch', 'flex-start', 'flex-end', 'center', 'baseline']}
      setSelectedValue={setAlignItems}>
      <View style={[styles.box, {backgroundColor: 'powderblue'}]} />
      <View style={[styles.box, {backgroundColor: 'skyblue'}]} />
      <View
        style={[
          styles.box,
          {
            backgroundColor: 'steelblue',
            width: 'auto',
            minWidth: 50,
          },
        ]}
      />
    </PreviewLayout>
```



f) Align Self

Właściwość "alignSelf" ma te same opcje i efekt co "alignItems", ale zamiast wpływać na dzieci wewnątrz kontenera, można zastosować tę właściwość do pojedynczego dziecka, aby zmienić jego wyrównanie w obrębie rodzica. "alignSelf" nadpisuje wszelkie opcje ustawione przez rodzica za pomocą "alignItems". Innymi słowy, pozwala to na indywidualne kontrolowanie wyrównania jednego dziecka wewnątrz kontenera, niezależnie od ogólnego wyrównania dzieci w kontenerze.

g) Align Content

Właściwość "alignContent" definiuje rozmieszczenie linii względem osi poprzecznej. Ma to znaczenie tylko wtedy, gdy elementy są zawijane na wiele linii przy użyciu "flexWrap".

- "flex-start" (wartość domyślna): Wyrównuje zawinięte linie do początku osi poprzecznej kontenera.

- "flex-end": Wyrównuje zawinięte linie do końca osi poprzecznej kontenera.

- "stretch" (wartość domyślna w przypadku korzystania z Yoga na stronie internetowej): Rozciąga zawinięte linie, aby dopasować się do wysokości osi poprzecznej kontenera.

- "center": Wyśrodkowuje zawinięte linie na środku osi poprzecznej kontenera.

- "space-between": Równomiernie rozmieszcza zawinięte linie wzdłuż osi poprzecznej kontenera, dzieląc pozostałą przestrzeń między liniami.

- "space-around": Równomiernie rozmieszcza zawinięte linie wzdłuż osi poprzecznej kontenera, dzieląc pozostałą przestrzeń wokół linii. W porównaniu do "space-between", użycie "space-around" spowoduje, że przestrzeń zostanie rozdzielona na początku pierwszej linii i na końcu ostatniej linii.

h) Flex Wrap

Właściwość "flexWrap" jest ustawiana na kontenerach i kontroluje, co się dzieje, gdy dzieci wychodzą poza rozmiar kontenera wzdłuż głównej osi. Domyślnie dzieci są zmuszane do zmieszczenia się w jednym wierszu (co może prowadzić do zmniejszenia elementów). Jeśli jest zezwolone na zawijanie, elementy są zawijane na wiele wierszy wzdłuż głównej osi, jeśli jest to konieczne.

4) Obrazki

React Native dostarcza jednolity sposób zarządzania obrazami i innymi zasobami multimedialnymi w aplikacjach na platformy Android i iOS. Aby dodać statyczny obraz do swojej aplikacji, umieść go w jakimś miejscu w drzewie kodu źródłowego i odwołuj się do niego w ten sposób:

```
<Image source={require('./my-icon.png')} />
```

Nazwa obrazu jest rozwiązywana w ten sam sposób, w jaki są rozwiązywane moduły JavaScript. W przykładzie powyżej bundler będzie szukał pliku my-icon.png w tym samym folderze, co komponent, który go wymaga.

Możesz używać przyrostków @2x i @3x, aby dostarczać obrazy o różnych gęstościach ekranu. Jeśli masz następującą strukturę plików:

```
.
├── button.js
└── img
    ├── check.png
    ├── check@2x.png
    └── check@3x.png
```

Bundler będzie pakować i dostarczać obraz odpowiadający gęstości ekranu urządzenia. Na przykład, plik check@2x.png zostanie użyty na iPhone 7, podczas gdy check@3x.png zostanie użyty na iPhone 7 Plus lub Nexusie 5. Jeśli nie ma obrazu odpowiadającego gęstości ekranu, zostanie wybrana najlepsza dostępna opcja.

Na systemie Windows może być konieczne ponowne uruchomienie bundlera po dodaniu nowych obrazów do projektu.

Oto kilka korzyści, jakie uzyskujesz:

1. Ten sam system na Androidzie i iOS.

2. Obrazy znajdują się w tym samym folderze co Twój kod JavaScript. Komponenty są samowystarczalne.

3. Brak globalnej przestrzeni nazw, czyli nie musisz się martwić o kolizje nazw.

4. Do aplikacji pakowane są tylko obrazy, które są faktycznie używane.

5. Dodawanie i zmienianie obrazów nie wymaga ponownej kompilacji aplikacji, możesz odświeżyć symulator tak, jak zwykle.

6. Bundler zna wymiary obrazów, więc nie trzeba ich duplikować w kodzie.

7. Obrazy mogą być dystrybuowane za pomocą pakietów npm.

Aby to działało, nazwa obrazu w funkcji "require" musi być znana statycznie.

```
// GOOD
<Image source={require('./my-icon.png')} />;

// BAD
const icon = this.props.active
  ? 'my-icon-active'
  : 'my-icon-inactive';
<Image source={require('./' + icon + '.png')} />;

// GOOD
const icon = this.props.active
  ? require('./my-icon-active.png')
  : require('./my-icon-inactive.png');
<Image source={icon} />;
```