

React Native od zera

Jakub Płoskonka, Kacper Sikora



Część Teoretyczna

Historia powstania

- Po nieudanej premierze mobilnej aplikacji Facebooka, napisanej w HTML, pojawiła się potrzeba stworzenia nowego środowiska.
- Celem Facebooka było przeniesienie wszystkich korzyści związanych z tworzeniem stron internetowych na urządzenia mobilne.

Czemu React Native

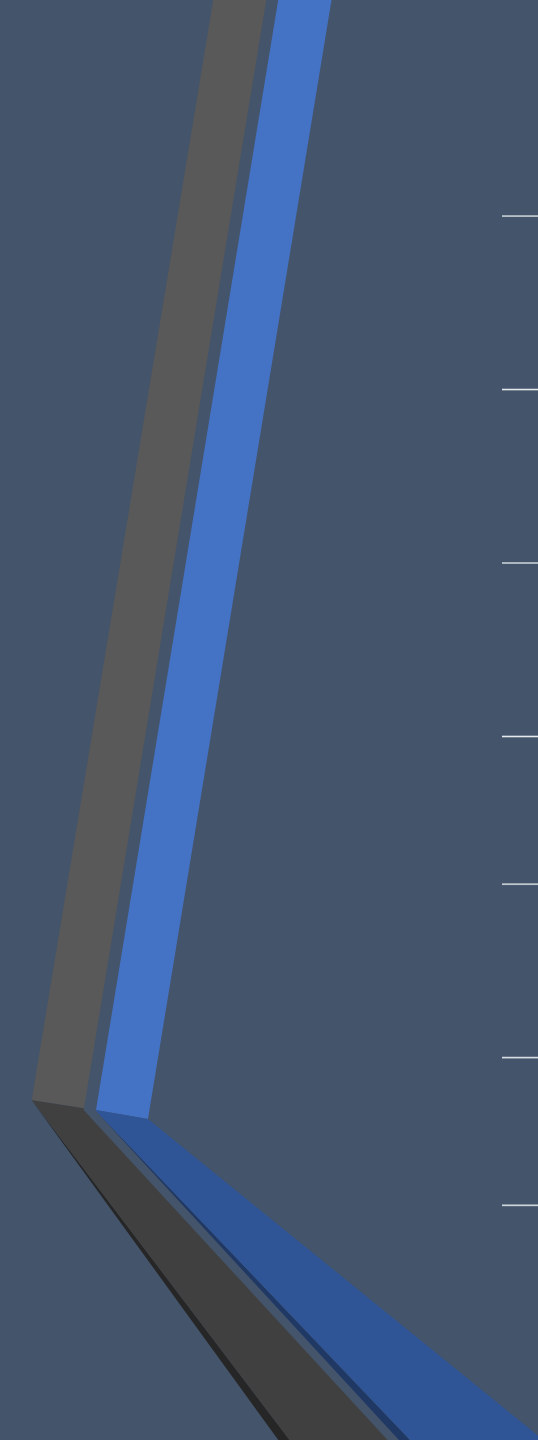
- Jeden kod na wiele platform, zapewniający spójne doświadczenie użytkownika
- Obniżenie czasu i kosztów wprowadzenia aplikacji na rynek

Używane technologie

- Zasady działania RN są praktycznie identyczne z React
- Komponenty RN otaczają istniejący kod natywny i wchodzi w interakcję z natywnymi interfejsami API
- RN ma podobną składnię do CSS, nie używa HTML ani CSS wchodzi w interakcję z natywnymi interfejsami API

Konkurencja

- Główną konkurencją dla RN jest Flutter, czyli SDK od Google
- Flutter używa języka Dart, przez co nauka będzie cięższa niż w przypadku RN



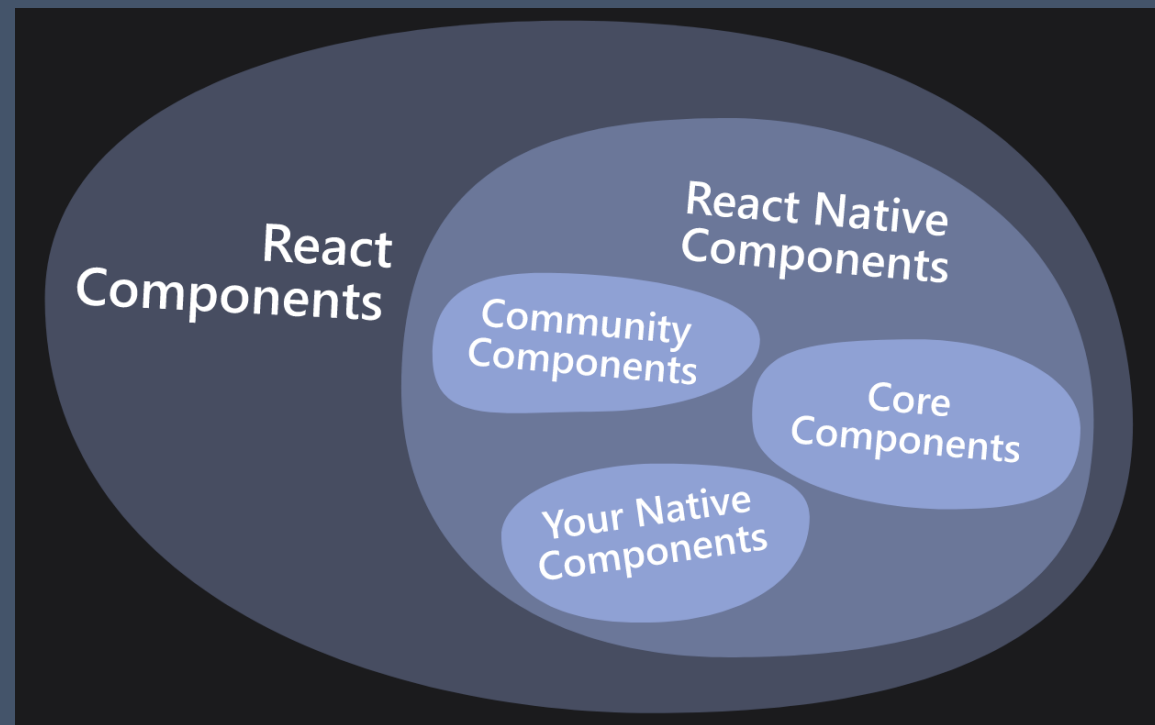
Feature	React Native	Flutter
Language	JavaScript is popular	Dart is a new language
User Interface	External UI kits, more UI options	Pre-built widgets, lesser UI customizations
Performance	Comparatively slower	Fast as it avoids bridges
Documentation	Disorganized	Simple and streamlined
Popularity	More popular and widely adopted	Less popular
Community Support	Large	Relatively small
Industry Trends	Better job prospects	Fewer jobs, bound to increase in the future



Początki nauki

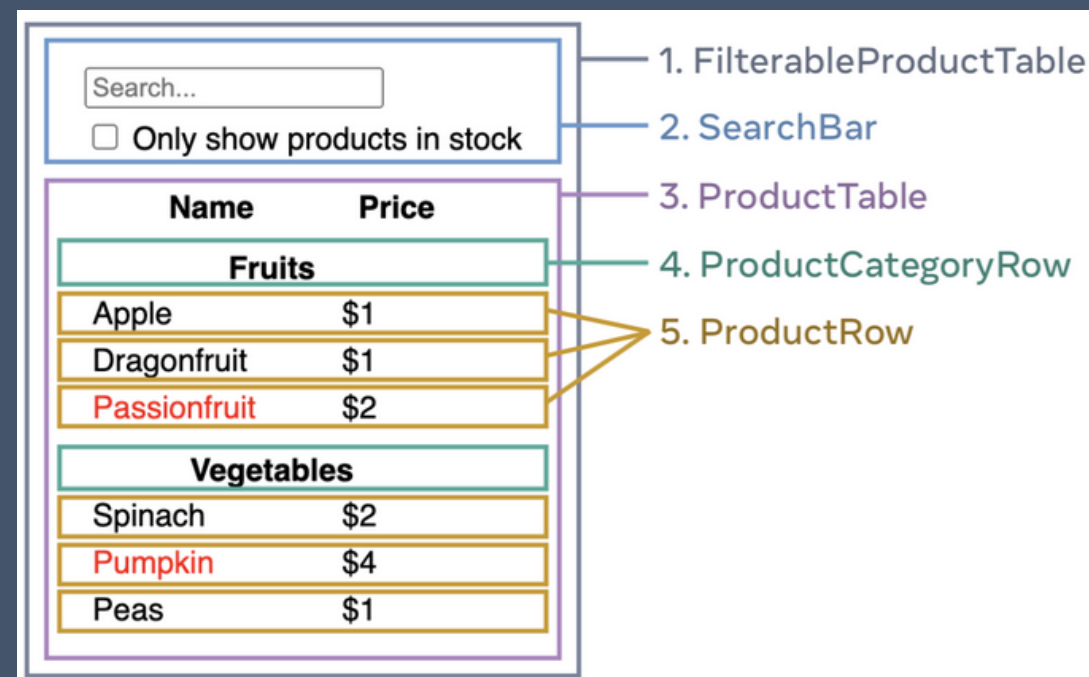
Jak zaczynaliśmy?

- Od nauki zwykłego Reacta
- Poznanie JSX, komponentów, stanów
- <https://react.dev/learn>



Hierarchia komponentów

- Tworzymy komponenty wielorazowego użytku
- Podstawową techniką jest przekazywanie właściwości do komponentów



Komponenty RN

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
<code><View></code>	<code><ViewGroup></code>	<code><UIView></code>	A non-scrolling <code><div></code>	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
<code><Text></code>	<code><TextView></code>	<code><UITextView></code>	<code><p></code>	Displays, styles, and nests strings of text and even handles touch events
<code><Image></code>	<code><ImageView></code>	<code><UIImageView></code>	<code></code>	Displays different types of images
<code><ScrollView></code>	<code><ScrollView></code>	<code><UIScrollView></code>	<code><div></code>	A generic scrolling container that can contain multiple components and views
<code><TextInput></code>	<code><EditText></code>	<code><UITextField></code>	<code><input type="text"></code>	Allows the user to enter text

Jak stworzyć niestandardowy komponent?

- Za pomocą podstawowych komponentów, możemy tworzyć własne, które będą wielokrotnego użytku

```
1  import React from 'react';
2  import {Text, View} from 'react-native';
3
4  const Cat = () => {
5    return (
6      <View>
7        <Text>Hello, I am cat!</Text>
8      </View>
9    );
10 };
11
12 export default Cat;
```

```
import React from "react";  
import Cat from "../Cat";
```

```
const App = () => {  
  return (  
    <View>  
      <Cat/>  
      <Cat/>  
      <Cat/>  
    </View>  
  );  
};
```

```
export default App;
```

I am your a cat!
I am your a cat!
I am your a cat!

Przekazywanie właściwości

- JSX to rozszerzenie składni języka JavaScript, które umożliwia pisanie znaczników przypominających HTML w pliku JavaScript
- <https://react.dev/learn/writing-markup-with-jsx>

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

const Cat = props => {
  return (
    <View>
      <Text>Hello, I am {props.name}!</Text>
    </View>
  );
};

const App = () => {
  return (
    <View>
      <Cat name="Maru" />
      <Cat name="Jellylorum" />
      <Cat name="Spot" />
    </View>
  );
};

export default App;
```

Przekazywanie właściwości

```
import React from "react";
import { Image } from "react-native";

export function getImageUrl(person, size = "s") {
  return "https://i.imgur.com/" + person.imageId + size + ".jpg";
}

function Avatar({ person, size }) {
  return (
    <Image
      className="avatar"
      src={getImageUrl(person)}
      alt={person.name}
      width={size}
      height={size}
    />
  );
}
```

```
export default function Profile({ person, size, isSepia, thickBorder }) {
  return (
    <View className="picture">
      <Avatar
        person={person}
        size={size}
        isSepia={isSepia}
        thickBorder={thickBorder}
      />
    </View>
  );
}
```

Listy na ekranie

FlatList

- Służy do wyświetlania listy elementów, które są renderowane za każdym razem kiedy pojawiają się na ekranie

ScrollView

- Zapamiętuje stan elementów, ale przez to trzyma wszystkie w pamięci RAM

FlatList

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

const data = [
  { key: "Martin" },
  { key: "Jerzy" },
  { key: "Donos" },
];

const FlatListBasics = () => {
  return (
    <View>
      <FlatList
        data={data}
        renderItem={({ item }) => <Text>{item.key}</Text>}
      />
    </View>
  );
};

export default FlatListBasics;
```

ScrollView

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

const logo = {
  uri: 'https://reactnative.dev/img/tiny_logo.png',
  width: 64,
  height: 64,
};

const ScrollViewBasic = () => (
  <ScrollView>
    <Text style={{fontSize: 96}}>Scroll me plz</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
  </ScrollView>
);

export default ScrollViewBasic;
```

useState

- React za każdym razem renderuje komponenty od zera oraz zmienne lokalne nie są zachowywane podczas ponownego renderowania
- Potrzebujemy użyć Hook'ów useState, które zapewnią nam brakującą funkcjonalność

```
import { React, useState } from "react";
import { Button } from "react-native";

const Clicker = () => {
  const [count, setCount] = useState(0);
  const onPress = () => setCount(count + 1);

  return <Button onPress={onPress} title={count} />;
};

export default Clicker;
```

TextInput

- Wprowadzanie tekstu, który będzie od razu wyświetlany, musi używać `useState`, ponieważ każdorazowo po wpisaniu znaku, komponent będzie aktualizowany

```
import React from "react";
import { useState } from "react";
import { SafeAreaView, TextInput } from "react-native";
import { styles } from "../styles";

const TextInputExample = () => {
  const [text, onChangeText] = useState("Useless Text");
  const [number, onChangeNumber] = React.useState("");

  return (
    <SafeAreaView>
      <TextInput
        style={styles.input}
        onChangeText={onChangeText}
        value={text}
      />
      <TextInput
        style={styles.input}
        onChangeText={onChangeNumber}
        value={number}
        placeholder="useless placeholder"
        keyboardType="numeric"
      />
    </SafeAreaView>
  );
};

export default TextInputExample;
```



Omówienie projektu

Struktura

- wip

Użyte komponenty

- wip

Rozwój

- wip



QUIZ



Część Praktyczna

Uruchomienie środowiska

- Oficjalna strona:
<https://reactnative.dev/docs/environment-setup>
- Instalacja Node.js
- Uruchomienie Expo
- Opcjonalnie Android Studio

Pierwsze zadanie

- stworzenie kontenera który będzie przechowywał komponenty
- stworzenie komponentu z tagiem text który będzie automatycznie wypełniany za pomocą tagu input (useState)

Drugie zadanie

- stworzenie pliku json z danymi (id, tytuł, image/icon)
- dodanie komponentu z flatlistą wykorzystującą plik json

Trzecie zadanie

- dodanie Tab nawigacji
- stylowanie nawigacji, dodawanie ikon

