infoShare ACADEMY

React

Komponenty, JSX, renderowanie kolekcji i warunkowe

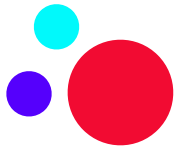
infoShare Academy

HELLO

Jakub Wojtach

Senior **full stack** developer

infoShare ACADEMY



Rozgrzewka i agenda

Zacznijmy ten dzień z przytupem!

Rozgrzewka

Dżem

Lady Pank

Rozgrzewka

Messi

Ronaldo

Rozgrzewka

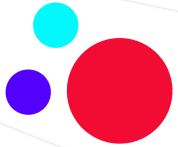
Ryż

Makaron

Rozgrzewka

Bal

Domówka

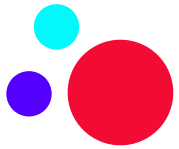


Agenda

- **Podstawy** teorii
- **Podstawy** biblioteki React z wykorzystaniem JSX
- **Renderowanie** kolekcji i renderowanie warunkowe
- **Pozostałe** zagadnienia i często popełniane błędy

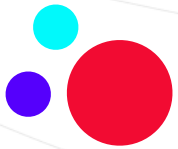
Współpraca

- Zadajemy pytania w dowolnym momencie - kanał **merytoryka**
- Krótkie przerwy (**5 min**) co godzinę
- Długa przerwa (**20 min**) po ostatnim bloku



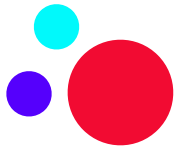
Podstawy teorii

infoShareAcademy.com



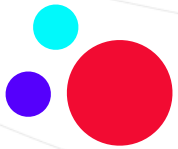
Podstawy teorii

- Będziemy korzystali z **Vite,** jako naszego preferowanego toolingu
- Na tych zajęciach **nie** rozmawiamy o **stanie (state)**
- Poznamy natomiast **props**
- Komponent to część składowa, którą wykorzystujemy przy budowaniu naszej aplikacji
- Nasza aplikacja, która zazwyczaj również jest komponentem składowym jest “instalowana” w root, czyli najczęściej divie z id “root”
- Renderowanie elementów
- Komponenty i propsy



React z wykorzystaniem JSX

infoShareAcademy.com



Element reactowy

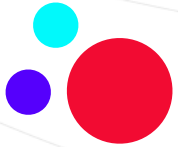
- Komponent to funkcja, która zwraca i wyświetla nam pewną część aplikacji. Tworzenie ich to podstawowa czynność w React. Wszystko sprowadza się do podzielenia aplikacji na małe komponenty.
- Dzielenie kodu na komponenty ułatwia pracę programiście oraz zmniejsza koszty utrzymania aplikacji.
- Tworząc re-używalne komponenty można wykorzystać je w innych częściach programu.
- Jeśli będzie potrzeba zmienić coś w danym komponencie to zmiana zostanie wykonana tylko jeden raz zamiast w X miejscach.
- Czas wykonania będzie krótszy, a programiście będzie łatwiej zapanować nad kodem

Element reactowy

Pierwsza aplikacja React – Kurs React – cz. 1

Luty 28, 2019 | Frontend, Kurs React | aplikacja, kurs-react, node, npm, react

W pierwszej części kursu przedstawiam jak stworzyć pierwszą aplikację React oraz opisuję podstawowe pojęcia z tym związane.



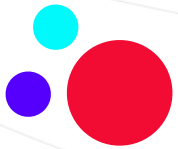
Definiowanie komponentów funkcyjnych

- Najprostszy sposób na zdefiniowanie komponentu.
- Piszemy je za pomocą javascriptowej funkcji

```
function Welcome(props) {
  return <h1>Cześć, {props.name}</h1>;
}
```

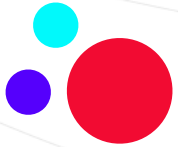
- Funkcja ta jest poprawnym reactowym komponentem, ponieważ przyjmuje pojedynczy argument “props” (*właściwości*, properties), będący obiektem z danymi
- Funkcja może być również wyrażeniem funkcyjnym z zachowaniem odpowiednich zasad, jak zwrócenie czegoś w kwerendzie **return**

Zadanie



Przekazywanie danych między komponentami

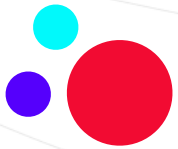
- Do przekazywania danych między komponentami służy nam argument **props**
- Props to po prostu properties, które w zapisie są podobne do atrybutów HTML
- Komponenty korzystające jedynie z props z założenia są “read-only”, czyli same w sobie nie zmieniają swojego **stanu**, jedynie operują na danych przesłanych z zewnątrz
- Dane te mogą być przesłane z innych komponentów, podobnie jak przekazywane są argumenty do funkcji JS
- Komponent może je otrzymać, ale **nie może ich zmieniać**



Przekazywanie danych między komponentami

- Aby odwołać się do propsów komponentu, możemy odwołać się do argumentu funkcji **props**, albo **zdestrukturyzować** konkretne elementy, jakie chcemy wyciągnąć w danym komponencie

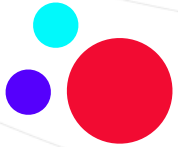
Zadanie



Renderowanie dynamicznych danych

- Komponent ma możliwość wygenerowania danych w sposób dynamiczny z użyciem propsów przekazanych z zewnątrz
- Propsem może być wartość, ale może to być również funkcja
- Stwórzmy zatem taki komponent, aby lepiej zrozumieć czym jest taka funkcja

Zadanie

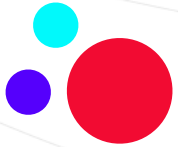


Kompozycja komponentów

- Komponent może mieć jedno, wiele lub wcale dzieci, czyli **children.**
- Children to treść przekazywana pomiędzy znacznikiem otwierającym i zamykającym dany komponent

```
<Profile>
<ProfileImage src="/asset/profile-img.png" />
<ProfileDetails name="Antonello" surname="Zanini" />
</Profile>
```

- Jest to **props**, który jest przekazywany w sposób automatyczny do każdego komponentu, co jest użyteczne w momencie, gdy chcemy wygenerować treść, która nie jest znana z założenia



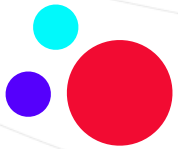
Kompozycja komponentów

- Komponent może mieć jedno, wiele lub wcale dzieci, czyli **children.**
- Children to treść przekazywana pomiędzy znacznikiem otwierającym i zamykającym dany komponent

```
<Profile>
<ProfileImage src="/asset/profile-img.png" />
<ProfileDetails name="Antonello" surname="Zanini" />
</Profile>
```

- Jest to **props**, który jest przekazywany w sposób automatyczny do każdego komponentu, co jest użyteczne w momencie, gdy chcemy wygenerować treść, która nie jest znana z założenia

Zadanie



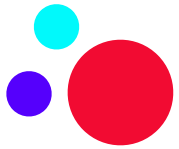
Stylowanie komponentów (inline)

- Komponenty w react można stylować na wiele różnych sposobów
- Jednym z nich jest stylowanie liniowe
- Style definiowane w sposób liniowy deklarujemy jako obiekt JS, a zapisujemy je w **podwójnych nawiasach**

```
<div>
  <h1 style={{color: "red"}}>Hello Style!</h1>
  <p>Add a little style!</p>
</div>
```

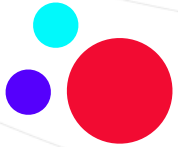
infoShare ACADEMY

Zadanie



Renderowanie kolekcji i warunkowe

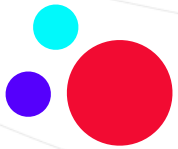
infoShareAcademy.com



Renderowanie kolekcji danych

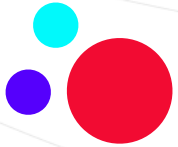
- W JSX możemy używać tablic, aby wyświetlać kolekcje elementów (np. listę użytkowników).
- Tablice mogą zawierać wszystkie typy danych, które mogą zostać wyrenderowane, to znaczy: liczy, łańcuchy tekstu, komponenty, elementy, null, undefined, wartości logiczne (true, false).
- Jeżeli zawierają obiekty, to nie będzie można ich wyrenderować, chyba że z obiektów zostaną wybrane konkretne właściwości (o typach "dopuszczalnych", jak wyżej wymienione).
- Tablice w Reakcie działają bardzo podobnie jak w "normalnych" funkcjach.

Zadanie wspólne



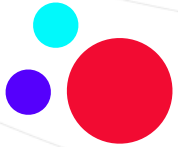
Property “key”

- https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b31bfa42



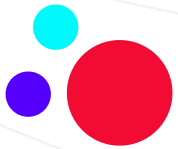
Renderowanie warunkowe

- React umożliwia tworzenie komponentów, które hermetyzują (ang. *encapsulate*) pożądane przez ciebie metody.
- Wybrane komponenty mogą być renderowane bądź nie, w zależności od stanu twojej aplikacji.
- Renderowanie warunkowe działa w React tak samo jak instrukcje warunkowe w JS.
- Aby stworzyć elementy odzwierciedlające aktualny stan aplikacji, należy użyć instrukcji if lub operatora warunkowego oraz pozwolić Reactowi je dopasować poprzez aktualizację interfejsu użytkownika.



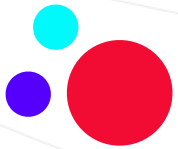
Renderowanie warunkowe

- React umożliwia tworzenie komponentów, które hermetyzują (ang. *encapsulate*) pożądane przez ciebie metody.
- Wybrane komponenty mogą być renderowane bądź nie, w zależności od stanu twojej aplikacji.
- Renderowanie warunkowe działa w React tak samo jak instrukcje warunkowe w JS.
- Aby stworzyć elementy odzwierciedlające aktualny stan aplikacji, należy użyć instrukcji if lub operatora warunkowego oraz pozwolić Reactowi je dopasować poprzez aktualizację interfejsu użytkownika.



Renderowanie warunkowe

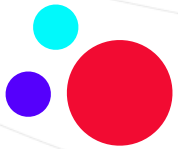
- if
- if else
- ternary
- &&



Renderowanie warunkowe if

- Najprostsza metoda renderowania warunkowego w react wykorzystuje pojedynczego **ifa**
- Przykładowe użycie to nie renderowanie danego komponentu w momencie, gdy nie przekazany jest jakiś properties
- Deklarujemy taką metodę jako dodatkowy if statement przed statementem **return**
- **Wewnątrz** ifa sprawdzamy warunek, po czym zwracamy coś co chcemy zwrócić, gdy dany warunek będzie spełniony

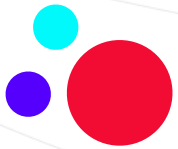
Zadanie



Renderowanie warunkowe if else

- Nieco bardziej złożona sytuacja, możemy dodać dodatkowy if, sprawdzający więcej warunków wewnątrz uprzednio dodanego komponentu, jak np. Sprawdzenie czy tablica elementów przekazanych jako jeden z propsów ma odpowiednią długość, a jeśli nie - wyświetlić tekst alternatywny. W innym wypadku **normalnie** wyrenderować komponent.
- Co ważne - gdy w else mamy zwykłe renderowanie - możemy go pominąć i zostawić tylko return. Jeśli jednak renderujemy coś innego - możemy zostawić to w else.

Zadanie



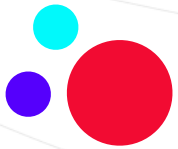
Renderowanie warunkowe ternary

- Jest to nieco inny zapis if/else, polegający na poniższym zapisie

```
// if else
function getFood(isVegetarian) {
  if (isVegetarian) {
    return 'tofu';
  } else {
    return 'fish';
  }
}
```

```
// ternary operator
function getFood(isVegetarian) {
  return isVegetarian ? 'tofu' : 'fish';
}
```

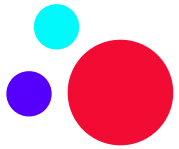
Zadanie



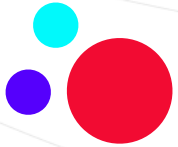
Renderowanie warunkowe &&

- Często zdarza się, że chcemy wyrenderować element w momencie gdy spełnia odpowiedni warunek, albo nie wygenerować go wcale
- Dobrym zastosowanie w takim momencie jest operator **&&**

```
function LoadingIndicator({ isLoading }) {
  return <div>{isLoading && <p>Loading ...</p>}</div>;
}
```

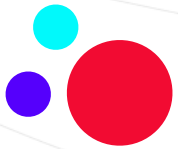


Pozostałe zagadnienia i często popełniane błędy



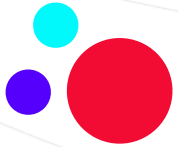
React fragment

- Jeden z elementów w API Reacta, stworzony po to by “obejść” konieczność zwracania pojedynczego elementu typu **parent** jako wyjściowy element komponentu
- Wyróżnia się tym, że daje możliwość zgrupowania listy potomków, bez konieczności dodawania zbędnych węzłów do drzewa DOM
- Przykłady i szersze wyjaśnienie tematu **https://pl.reactjs.org/docs/fragments.html**



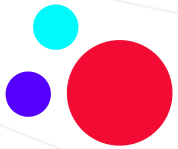
Uwzględnienie () po słowie **return**

- Return jest funkcją, która ma za zadanie **render** tego, co ma być wyrysowane na ekranie w ramach danego komponentu.
- Jeśli nasz komponent nie robi niczego poza renderingiem możemy zapisać taki komponent jako arrow function i pominąć brackety.
- Jeśli jednak robimy coś poza renderowaniem - należy pamiętać o dodaniu ich



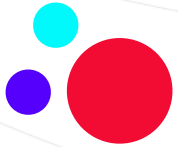
Często popełniane błędy

- **Zagnieżdżanie definicji komponentów** - często w ramach jednego komponentu błędnie deklarujemy kolejne z nich w tym samym pliku zamiast wyodrębnić je na zewnątrz.
- W momencie gdy komponent posiada dodatkową logikę wewnątrz jest to błędne
- Jest to błędne w momencie, gdy ten komponent moglibyśmy wykorzystać więcej niż raz poza tym komponentem.
- W takim wypadku należy stworzyć osobny komponent i wykorzystać go jako zaimportowany element w naszym komponencie



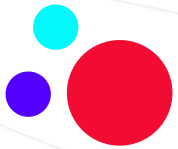
Często popełniane błędy

- **Zapominanie o użyciu key** - aby zachować szybkość i dynamikę silnika React należy pamiętać o korzystaniu z mechanizmu key.
- Jest to ważne properties, bo dzięki niemu VirtualDOM wie jaki element listy się zmienił i zaktualizuje jedynie ten konkretny element
- W momencie gdy brakuje key - lista jest **generowana od nowa**



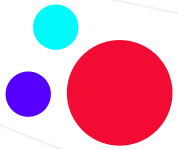
Często popełniane błędy

- **Próba zwrócenia wielu elementów React z komponentu** - jak było wspomniane wcześniej React z założenia zmusza nas do zmuszenia pojedynczego rodzica
- Rozwiązaniem takiego problemu może być np. Zwrócenie kilku elementów jako tablica, albo opakowanie ich w fragment lub dodatkowy kontener



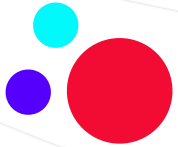
Często popełniane błędy

- **Pomijanie PascalCase przy definicji komponentów** - w ramach konwencji przyjętej podczas tworzenia komponentów założono, iż by odróżnić “customowe” elementy Reactowe od tych HTML’owych zapisujemy je jako **PascalCase**.
- Dzięki temu przeglądarka nie zwróci nam błędu, gdyż silnik React będzie wiedział, że odnosimy się do komponentu stworzonego w tym frameworku, a nie **niezidentyfikowanego** elementu HTML



Często popełniane błędy

- **Class zamiast className** - jest to jeden z przykładów propertiesów HTML, które zostały dostosowane do wykorzystania wewnątrz komponentów.
- Z racji na to, że JSX jest rozszerzeniem JS - nie może on korzystać z propertiesów, które mają szczególne znaczenie w samym języku.
- Gdy korzystamy z **className** React wie, iż próbujemy się odnieść do klasy elementu, a nie do zarezerwowanego propertiesu JS
- Więcej o tym w linkach dla chętnych



Ciekawe linki

- https://www.robinwieruch.de/react-function-component/
- https://codeburst.io/a-complete-guide-to-props-children-in-react-c315fab74e7c
- https://kentcdodds.com/blog/understanding-reacts-key-prop
- https://www.robinwieruch.de/conditional-rendering-react/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords

Dziękuję za uwagę

Jakub Wojtach