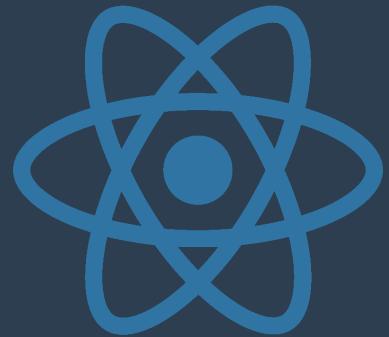




REACT

57 PYTAŃ REKRUTACYJNYCH



**Poznaj najpopularniejsze pytania
rekrutacyjne z Reacta i zdobądź
lepszą pracę**

Patryk Omiotek

Wstęp

Cześć!

Niezwyczajne mi miło, że korzystasz z tego e-booka. Mam nadzieję, że pomoże Ci zmienić pracę na lepszą lub zdobyć pierwszą pracę w IT :)

Dokument powstał, aby podzielić się wiedzą dotyczącą pytań technicznych, jakie możesz usłyszeć w trakcie rekrutacji. Do niektórych z odpowiedzi przygotowałem kod, który możesz przeanalizować. Część odpowiedzi jest krótka, ale za to zwięzła i dosadnie wyczerpuje temat.

Oczywiście informacje zdobyte za pomocą tego e-booka nie zagwarantują Ci od razu nowej pracy, ale w znacznym stopniu pomogą usystematyzować dotychczasową wiedzę. Zachęcam gorąco, aby do każdego z pytań napisać samemu kilka linijek kodu, co pozwoli w wystarczającym stopniu na przyswojenie materiału.

O autorze



Nazywam się Patryk Omiotek, jestem programistą JavaScript, Python, PHP, trochę DevOpsem, trenerem IT oraz tzw. weryfikatorem IT, czyli osobą, która sprawdza techniczne umiejętności kandydatów ubiegających się na stanowiska junior, mid i senior developerów.

W zeszłym roku uruchomiłem projekt [Szkoła Reacta](#), gdzie znajdziesz praktyczne tutoriale jak korzystać z... Reacta :)

Spis treści

| | |
|---|----|
| • Czym jest React?..... | 6 |
| • Czym jest JSX?..... | 7 |
| • Czym są propsy?..... | 8 |
| • Czym jest stan komponentu?..... | 9 |
| • Jaka jest różnica pomiędzy state a props?..... | 10 |
| • Czym są komponenty "Pure"?..... | 11 |
| • Do czego służy drugi argument funkcji setState?..... | 12 |
| • Czy w JSX można stosować wyrażenia warunkowe?..... | 13 |
| • Czym jest Virtual DOM?..... | 14 |
| • Czym jest props "key" i do czego służy?..... | 15 |
| • Jakie są cykle życia komponentu?..... | 16 |
| • Jakie są lifecycle methods?..... | 19 |
| • Jakie są lifecycle methods zakazane w wersji 16?..... | 20 |
| • Kiedy korzystać z komponentów klasowych zamiast komponentów funkcyjnych?..... | 21 |
| • Jaka jest wymagana metoda dla komponentu klasowego?..... | 22 |
| • Jakie jest główne zadanie konstruktora w komponencie?..... | 23 |
| • Czy jest konieczne zdefiniowanie konstruktora dla komponentu?..... | 24 |
| • Jak przekazać event handler do komponentu?..... | 25 |
| • Jak ustawić stan, gdy klucz jest dynamiczny?..... | 26 |
| • Czym są komponenty kontrolowane (controlled components)?..... | 28 |
| • Czym są komponenty niekontrolowane (uncontrolled components)?..... | 29 |
| • Czy dobrze jest korzystać z „arrow functions” w metodach renderujących?..... | 31 |
| • Czym jest komponent wyższego rzędu (Higher-Order Component)?..... | 32 |
| • Czym jest props children?..... | 33 |
| • Jak przekazać liczby do komponentów?..... | 34 |
| • Dlaczego do stylowania React używa atrybutu className zamiast class?..... | 35 |

Spis treści

| | |
|--|----|
| • Czym są fragmenty?..... | 36 |
| • Dlaczego fragmenty są lepsze niż kontenery div?..... | 37 |
| • Czym są komponenty bezstanowe?..... | 38 |
| • Czym są komponenty stanowe (stateful components)?..... | 39 |
| • Jakie jest zastosowanie metody render w react-dom?..... | 40 |
| • Jak stylować komponenty?..... | 41 |
| • Jak korzystać z dekoratorów?..... | 42 |
| • Dlaczego należy nazywać nazwy komponenty z dużych liter?..... | 43 |
| • Jak można zmusić component do przerenderowania bez korzystania z setState?..... | 44 |
| • Jak zrobić pętlę w JSX?..... | 45 |
| • Jaka jest różnica pomiędzy React a ReactDOM?..... | 46 |
| • Dlaczego ReactDOM jest oddzielony od Reacta?..... | 47 |
| • Dlaczego nie możesz zmienić wartości propsów?..... | 48 |
| • Jak aktualizować component co sekundę?..... | 49 |
| • Jak sfocusować element input po załadowaniu strony?..... | 50 |
| • Jak zdefiniować stałe?..... | 51 |
| • Jak wywołać kliknięcie?..... | 52 |
| • Czy mogę dispatchować akcję w reducerze?..... | 53 |
| • Czy muszę przechowywać cały stan komponentu w Reduksie?..... | 54 |
| • Jak dispatchować akcję w trakcie ładowania aplikacji?..... | 55 |
| • Jaka jest różnica pomiędzy komponentem a kontenerem?..... | 56 |
| • W jaki sposób można zapisać mapDispatchToProps()?..... | 57 |
| • Jakie jest zastosowanie parametru ownProps w mapStateToProps oraz mapDispatchToProps?..... | 58 |
| • Jak zaprojektować strukturę aplikacji, która korzysta z Reduksa?..... | 59 |
| • Czym jest akcja w Reduksie?..... | 60 |
| • Czy Redux może być wykorzystywany tylko z Reactem?..... | 61 |

Spis treści

| | |
|---|----|
| • Jakie jest zastosowanie registerServiceWorker?..... | 62 |
| • Czym są Hooki w Reakcie?..... | 63 |
| • Jakie są zasady stosowania Hooków?..... | 65 |
| • Czym jest funkcja lazy?..... | 66 |
| • Bonusy..... | 67 |

Pytanie 1

Czym jest React?

Jest biblioteką do renderowania interfejsów użytkownika. To wszystko - React nie jest frameworkm, czyli całym zestawem rozwiązań przygotowanych do zbudowania aplikacji. Za jego pomocą budujemy interfejs, czyli warstwę prezentacji dla aplikacji webowych i mobilnych.

To jest właśnie główne zadanie Reacta. Co jednak z widokami, komunikacją z API, stylowaniem aplikacji? Większość z tych rzeczy programujemy, ale nie bezpośrednio wykorzystując Reacta, a mamy tutaj dowolność wykorzystania *fetch*, *axios*, *Styled-components*, *CSS*, *Sass* itd.

Poznając Reacta musisz skupić na tym jak tworzyć komponenty, jak zaprojektować strukturę plików i katalogów oraz jak powinna wyglądać kompozycja komponentów.

Pytanie 2

Czym jest JSX?

JSX (akronim od *JavaScript XML*) jest rozszerzeniem *ECMAScript* i służy do budowania komponentów za pomocą połączenia *HTML* (widok) i JS (logika). Na pierwszy rzut oka to połączenie może wydawać się co najmniej dziwne, ale znacznie przyspiesza pracę w porównaniu do pisania komponentów za pomocą *React.createElement()*.

W poniższym przykładzie funkcja render zwraca widok prostego komponentu. Co ważne, w widokach możemy wykorzystywać zmienne oraz wykonywać instrukcje JavaScript. Instrukcje należy zatrzymać pomiędzy klamerkami tj. “{zmienna/instrukcja}”.

```
class App extends React.Component {
  render() {
    return(
      <div>
        <button>Hello world!</button>
        <p>One plus two equals: {1 + 2}</p>
      </div>
    )
  }
}
```

Pytanie 3

Czym są propsy?

Jako propsy możemy rozumieć dane wejściowe, które przyjmuje komponent. Propsy mogą być typu prostego lub złożonego (tablice, obiekty). Przekazujemy je w sposób podobny do atrybutów w *HTML*, ale pamiętając o przepływie “z góry na dół” czyli z komponentu na wyższym poziomie, do komponentów-dzieci.

Główne cele propsów w Reakcie, to dostarczanie funkcjonalności takich jak:

1. Przekazywanie danych do komponentów
2. Wywoływanie zmian stanu przez handlery (np. onClick dla przycisków)
3. Korzystanie z ich wartości w metodach komponentu lub metodzie render za pomocą *this.props.nazwaPropsa*.

Dla przykładu, jeśli w aplikacji mamy komponent Button oraz chcemy mu ustawić kolor tła na czerwony, możemy to zrobić za pomocą:

```
<Button color="red" />
```

Natomiast “color” staje się nazwą naszego propsa. Będziemy wtedy mogli odwołać się do jej wartości za pomocą: *this.props.color*.

O propsach możemy też myśleć jako o parametrach przekazywanych do funkcji.

Pytanie 4

Czym jest stan komponentu?

Stan komponentu jest obiektem, który przechowuje informacje zmieniające się w trakcie życia komponentu. Zawsze powinniśmy starać się, aby nasz stan był jak najprostszy, a w aplikacji dążyć do posiadania jak największej liczby komponentów, które nie posiadają stanu. Przykład z komponentem `Message`:

```
class Message extends React.Component {  
  
  constructor(props) {  
    super(props)  
  
    this.state = {  
      text: 'Hello world'  
    }  
  }  
  
  render() {  
    return (  
      <div>  
        <p>{this.state.text}</p>  
      </div>  
    )  
  }  
}
```

Stan jest podobny do propsów, ale jest prywatny i w pełni kontrolowany przez komponent tzn. niedostępny dla żadnego innego komponentu.

Pytanie 5

Jaka jest różnica pomiędzy state a props?

Propsy oraz stan posiadają cechy wspólne: są zwykłymi obiektami *JavaScript*, oraz przechowują informacje, które wpływają na wynik funkcji renderującej. Natomiast różnią się pomiędzy sobą funkcjonalnością. Propsy przekazywane są do komponentu podobnie jak atrybuty do tagów *HTML* i posiadają wartości, natomiast stan jest zarządzany wewnątrz komponentu i nie można go przekazać do komponentu.

Przykładem komponentu, który ma swój wewnętrzny stan jest element *input*. Za każdym razem, gdy wprowadzasz do pola tekstowego nowy znak, zarazem zmienia się stan wewnętrzny oraz wartość tego pola.

Pytanie 6

Czym są komponenty typu "Pure"?

React.PureComponent jest właściwie tym samym, co `React.Component` z takim wyjątkiem, że automatycznie implementuje metodę `shouldComponentUpdate()`. Gdy propsy lub stan komponentu się zmieniają, PureComponent zrobi porównanie zmian wartości zarówno na stanie jak i na propsach. React.Component z drugiej strony nie porówna aktualnych propsów oraz stanu z nowymi wartościami. W ten sposób komponent domyślnie przerenderuje się za każdym razem, gdy `shouldComponentUpdate()` zostanie wywołane.

Pytanie 7

Do czego służy drugi argument funkcji setState?

Jest to *callback* wykonywany, gdy *setState* zakończy swoje działanie, czyli ustawi nowy stan komponentu. Należy pamiętać, że *setState* jest funkcją asynchroniczną, więc aby upewnić się, że korzystamy z aktualnego stanu, warto zawrzeć instrukcje w funkcji *callback*.

Przykład:

```
setState({ counter: value + 1}, () => {
  /* tutaj mamy blok, w którym na pewno stan jest już z
  aktualizowany nową wartością i możemy czytać: */
  console.log(this.state.counter)
})
```

Pytanie 8

Czy w JSX można stosować wyrażenia warunkowe?

Tak, można. Co prawda nie zrobimy instrukcji if, ale możemy to zrealizować wyrażenie warunkowe za pomocą operatora trójargumentowego:

```
<h1>Hello!</h1>
{
  (errors.length > 0) ?
    <h2>
      You have {errors.length} errors.
    </h2>
    :
    <h2>
      You don't have any errors.
    </h2>
}
```

Pytanie 9

Czym jest Virtual DOM?

Virtual DOM (VDOM) jest reprezentacją drzewa *DOM* w pamięci przeglądarki. Reprezentacja UI jest przechowywana w pamięci i synchronizowane są tylko aktualnie zmieniane elementy, a nie cała struktura *DOM*. To etap, który zachodzi pomiędzy wywołaniem funkcji renderującej, a wyświetlaniem elementów na ekranie.

Pytanie 10

Czym jest props “key” i do czego służy?

Props key jest specjalnym propsem, z którego należy korzystać, kiedy operujemy na tablicy elementów lub po prostu robimy iteracje. Propsy key pomagają Reactowi zidentyfikować w *Virtual DOM*, elementy które się zmieniły, zostały dodane lub usunięte.

Bardzo często korzysta się z identyfikatorów jako kluczy:

```
const taskElements = tasks.map(task =>
  <li key={task.id}>
    {task.title}
  </li>
)
```

Nie jest to jednak najlepsza praktyka, ponieważ takie identyfikatory mogą się powtórzyć. Znacznie bezpieczniej dodać prefiks do identyfikatora.

Bardzo często wykorzystuje się również indeksy iteracji jako klucze, ale takie rozwiązanie jeszcze w większym stopniu powoduje możliwość powielenia tych samych kluczy, a co za tym idzie React nie będzie dokładnie wiedział o który element chodzi w drzewie *Virtual DOM*.

```
const tasksElements = tasks.map((task, index) =>
  <li key={index}>
    {task.text}
  </li>
)
```

Pytanie 11

Jakie są cykle życia komponentu?

Każdy komponent posiada trzy etapy:

- Montowanie – komponent jest gotowy do zamontowania w drzewie DOM. W tym etapie wykonywane są metody `constructor()`, `getDerivedStateFromProps()`, `render()` oraz `componentDidMount()`
- Aktualizacja – w tym etapie komponent może zaktualizować się w trzech przypadkach – gdy dostanie nowe propsy, zmieni się stan poprzez wywołanie `setState()` lub zostanie wywołana metoda `forceUpdate()`. W tej metodzie wywoływane są metody `getDerivedStateFromProps()`, `shouldComponentUpdate()`, `render()`, `getSnapshotBeforeUpdate()` oraz `componentDidUpdate()`.
- Odmontowywanie – w tym etapie komponent nie jest już dłużej potrzebny i jest usuwany z drzewa DOM, dodatkowo jest wykonywana metoda `componentWillUnmount()`.

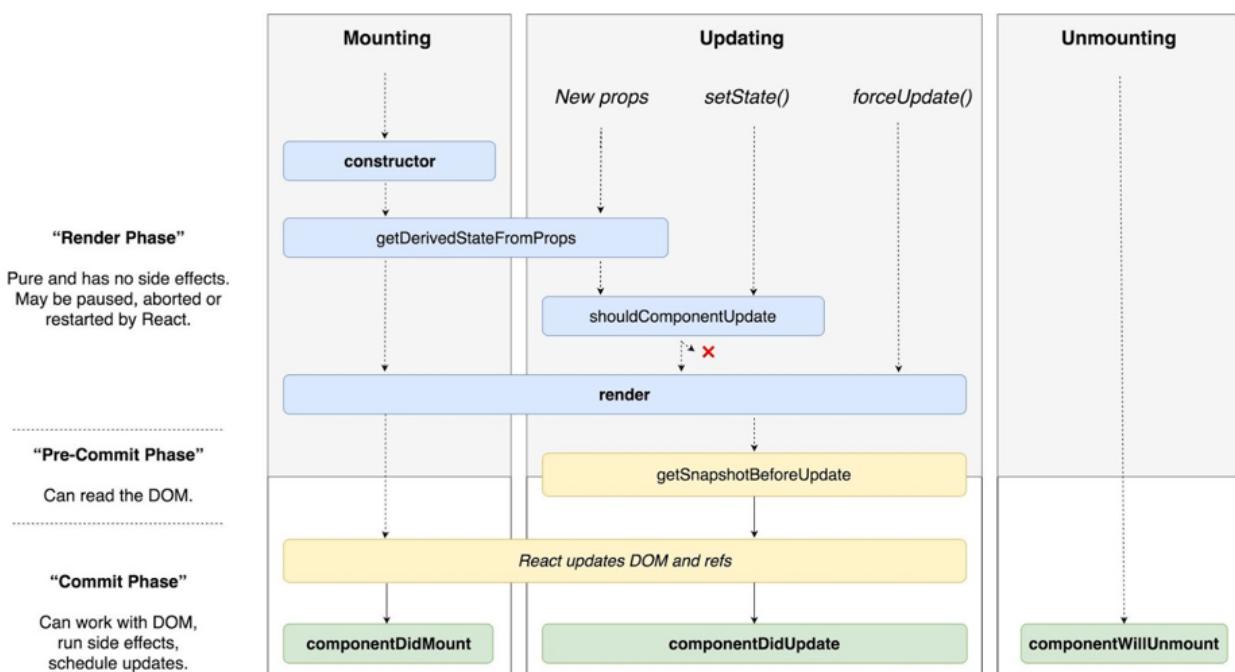
Warto wspomnieć, że wewnątrz Reacta zachodzą fazy dokonywania zmian w drzewie DOM. Dzielą się na etapy:

- Render – komponent wyrenderuje się bez efektów ubocznych. Zachodzi to dla komponentów typu Pure i React może zatrzymać, anulować lub zrestartować renderowanie.
- Pre-commit: zanim komponent zostanie zamontowany w drzewie DOM, istnieje proces który pozwala przeczytać Reactowi drzewo DOM za pomocą `getSnapshotBeforeUpdate()`
- Commit – React działa na drzewie DOM i wykonuje ostateczne metody: `componentDidMount()` po zamontowaniu, `componentDidUpdate()` po aktualizacji komponentu oraz `componentWillUnmount()` po odmontowaniu komponentu.

Pytanie 11

Jakie są cykle życia komponentu?

Etapy w React 16.3+:

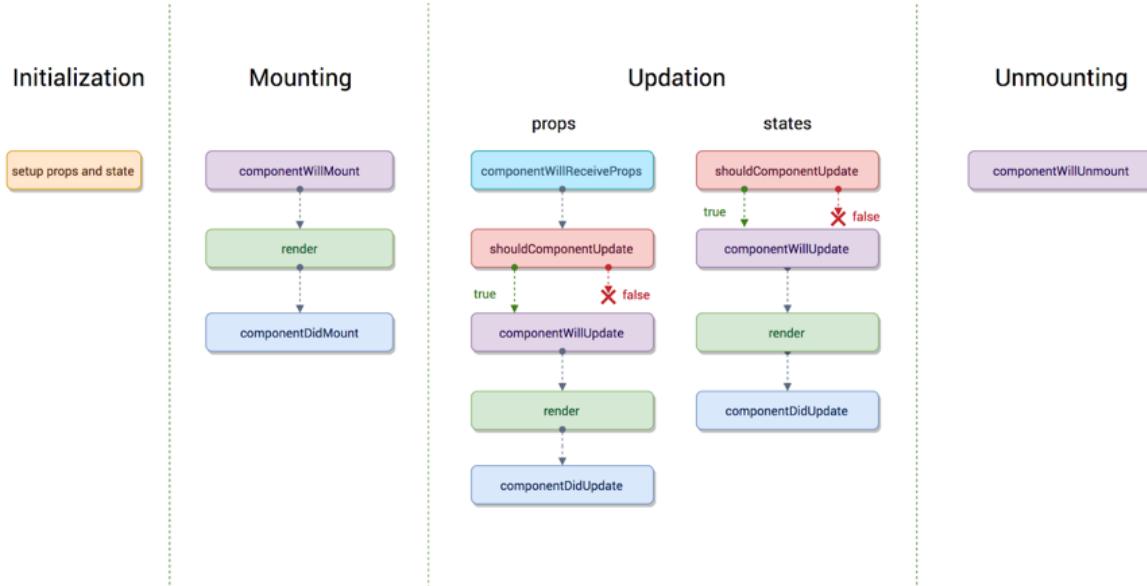


źródło: <http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

Pytanie 11

Jakie są cykle życia komponentu?

Wersje React przed 16.3:



źródło: <http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

Pytanie 12

Jakie są lifecycle methods?

React 16.3+

- *getDerivedStateFromProps* – wywoływana zaraz przed *render()* i wykonywana przy każdym przerenderowaniu komponentu.
- *componentDidMount* – wykonywana po pierwszym renderowaniu, można tutaj wykonywać instrukcje asynchroniczne, np. pobierać dane z serwera, zmiany stanu czy ustawiać listenery.
- *shouldComponentUpdate* – określa, czy komponent będzie zaktualizowany czy też nie. Domyślnie zwraca wartość *true*. Jeśli masz pewność, że komponent przy określonych warunkach nie powinien się przerenderować (przy określonych propsach czy stanie), możesz zaimplementować tę metodę zwracając jako wynik *false*. To sprawdza się głównie wtedy, gdy chcemy zadbać o wydajność aplikacji.
- *getSnapshotBeforeUpdate* – wykonywane zaraz przed tym, gdy komponent zostanie osadzony w drzewie DOM. Wartości zwracane przez tę metodę zostaną przekazane do *componentDidUpdate()*
- *componentDidUpdate* – przeważnie wykorzystywane do aktualizacji DOM w zależności od zmian stanu czy propsów. Metoda nie wykona się jeśli *shouldComponentUpdate()* zwróci *false*.
- *componentWillUnmount* – wykorzystywane do anulowania wykonywania requestów do API, usuwania listenerów, przerywania subskrypcji.

Pytanie 13

Jakie są lifecycle methods zakazane w wersji 16?

Poniższe metody stały się niezalecane w wykorzystaniu, zwłaszcza przy renderowaniu z użyciem funkcji asynchronicznych:

- `componentWillMount()`
- `componentWillReceiveProps()`
- `componentWillUpdate()`

Od React 16.3 te metody są poprzedzone prefiksem *UNSAFE_* i będą usunięte w wersji 17.

Pytanie 14

Jakie są lifecycle methods?

Przed React 16.3:

- *componentWillMount* – wykonywane przed renderowaniem i wykorzystywane do wprowadzania konfiguracji do komponentu
- *componentDidMount* – działa tak samo jak w 16.3+
- *componentWillReceiveProps* – wykonywane, gdy następują zmiany wartości propsów lub stanu
- *shouldComponentUpdate* – działa tak samo jak w 16.3+
- *componentWillUpdate* – wykonywane, przed przerenderowaniem komponentu, gdy *shouldComponentUpdate* zwróci true.
- *componentDidUpdate* – działa tak samo jak w 16.3+
- *componentWillUnmount* – działa tak samo jak w 16.3+

Pytanie 15

Kiedy korzystać z komponentów klasowych zamiast funkcyjnych?

Gdy komponent potrzebuje swojego wewnętrznego stanu lub lifecycle methods, wtedy powinniśmy korzystać z komponentów klasowych.

Tips: Od momentu wprowadzenia React Hooks można korzystać ze stanu komponentu również w komponentach funkcyjnych.

Pytanie 16

Jaka jest wymagana metoda dla komponentu klasowego?

Metoda `render()` jest jedyną wymaganą metodą dla komponentów klasowych. Wszystkie pozostałe są opcjonalne.

Pytanie 17

Jakie jest główne zadanie konstruktora w komponencie?

Konstruktor w komponencie jest wykorzystywany głównie do dwóch celów:

- Inicjalizowania lokalnego stanu komponentu poprzez przypisanie wartości do `this.state`
- Bindowania event handlerów do metod

Przykład:

```
constructor(props) {  
  super(props);  
  this.state = { counter: 0 }  
  this.handleClick = this.handleClick.bind(this)  
}
```

Pytanie 18

Czy jest konieczne definiowanie konstruktora dla komponentu?

Nie, tym bardziej, że w aplikacji możemy posiadać komponenty klasowe i funkcyjne. Nawet w pierwszym wypadku klasa nie musi posiadać konstruktora, gdy nie ma potrzeby ustawiania stanu czy bindowania metod.

Pytanie 19

Jak przekazać event handler do komponentu?

Możesz przekazać event handler jak i inne funkcje jako propsy do komponentów-dzieci w sposób:

```
<button onClick={this.handleClick}>
```

Ważne, aby nie wykonywać kodu w sposób:

```
<button onClick={this.handleClick()}>
```

Dlaczego? Ponieważ do spowoduje wykonanie handlера za każdym razem, co z kolei spowoduje zapętlenie aplikacji.

Pytanie 20

Jak ustawić stan, gdy klucz jest dynamiczny?

Taka sytuacja zdarza się np. gdy operujemy na formularzach i chcemy w stanie ustawić wartości z tego formularza. Zamiast tworzyć osobne metody na każde pole (wyobraź sobie, że masz formularz z 25 polami - sic!), wówczas możemy skorzystać z dynamicznego ustawiania stanu dla każdego z pól. Jeśli korzystasz z *ES6* lub *Babel.js* to zmiany swojego *JSX*, możesz skorzystać z tzw. *computed property names*:

```
handleChange(event) {
  this.setState({[event.target.id]: event.target.value })
}
```

Pytanie 21

Czym są komponenty kontrolowane (controlled components)?

Komponent, który kontroluje np. elementy typu input w formularzach jest nazywany komponentem kontrolowanym. Ponieważ każda zmiana wartości w polu (każde wprowadzenie nowego znaku) powoduje zmianę wartości, a ta odbywa się przez handler function. W przypadku gdy np. chcemy za każdym razem zamieniać znaki na duże litery, możemy skorzystać z kodu:

```
handleChange(event) {  
  this.setState({value: event.target.value.toUpperCase()})  
}
```

Pytanie 22

Czym są komponenty niekontrolowane (uncontrolled components)?

Komponenty niekontrolowane są takimi, które przechowują swój stan wewnętrznie i należy odwołać się do nich za pomocą drzewa *DOM* używając referencji, aby znaleźć obecną wartość. Jest to zbliżone do tradycyjnego *HTML'a*.

W poniższym kodzie komponent *UserProfile*, do pola input o nazwie name odwołujemy się za pomocą ref'a:

```
class UserProfile extends React.Component {  
  constructor(props) {  
    super(props)  
    this.handleSubmit = this.handleSubmit.bind(this)  
    this.input = React.createRef()  
  }  
  
  handleSubmit(event) {  
    alert('Current value: ' + this.input.current.value)  
    event.preventDefault()  
  }  
  
  render() {  
    return (  
      <form onSubmit={this.handleSubmit}>  
        <label>  
          Name: <input type="text" ref={this.input} />  
        </label>  
        <input type="submit" value="Submit" />  
      </form>  
    )  
  }  
}
```

Pytanie 22

Czym są komponenty niekontrolowane (uncontrolled components)?

W większości przypadków zalecane jest korzystanie z komponentów kontrolowanych, natomiast korzystanie z komponentów niekontrolowanych zachodzi wtedy, gdy korzystamy z dodatkowych bibliotek i nie da się tego zrobić w inny sposób.

Pytanie 23

Czy dobrze jest korzystać z „arrow functions” w metodach renderujących?

Tak, jak najbardziej. To najszybsza droga, aby przekazać parametry do funkcji *callback*. Ale należy pamiętać o optymalizacji wydajności w trakcie korzystania z tych funkcji.

```
class Button extends React.Component {  
  
  handleClick() {  
    console.log('Clicked!');  
  }  
  
  render() {  
    return (  
      <button onClick={() => this.handleClick()}>  
        Click me  
      </button>  
    )  
  }  
}
```

Tips: wykorzystywanie „arrow functions” w metodach renderujących za każdym razem tworzy w pamięci nową funkcję, gdy komponent renderuje się na nowo - to może spowodować spadek wydajności.

Pytanie 24

Czym jest komponent wyższego rzędu (Higher-Order Component)?

Komponent wyższego rzędu (*HOC*) jest funkcją, która przyjmuje dany komponent i zwraca nowy komponent dekorując go nowymi propsami. Jest to wzorzec, który naturalnie występuje w kompozycyjnej naturze Reacta. Komponenty z niego korzystające często nazywane *pure komponentami*, ponieważ akceptują każdy dynamicznie wprowadzony komponent-dziecko ale nie zmodyfikują lub skopiują ich zachowania względem wprowadzanych komponentów.

Najbardziej znane *HOC* to *connect* z pakietu *react-redux*, oraz wykorzystanie *react-router*, komponent *Route*.

```
const EnhancedComponent =  
  higherOrderComponent(WrappedComponent)
```

Pytanie 25

Czym jest props children?

Dzieci komponentu są dostępne w propsie children co pozwala na przekazanie komponentów jako dane wejściowe do innych komponentów. Dzieci umieszcza się pomiędzy otwierającym i zamykającym tagiem komponentu.

Przykład:

```
const MyComponent = React.createClass({
  render: function() {
    return <div>{this.props.children}</div>
  }
})

ReactDOM.render(
  <MyComponent>
    <span>Hello</span>
    <span>World</span>
  </MyComponent>,
  node
)
```

Pytanie 26

Jak przekazać liczby do komponentów?

Liczby powinniśmy przekazywać korzystając z klamer {}, natomiast stringi w cudzysłowach:

```
React.render(  
  <User age={34} department="IT" />,  
  document.getElementById('container')  
)
```

Pytanie 27

Dlaczego do stylowania React używa atrybutu `className` zamiast `class`?

Słowo kluczowe `class` należy do języka `JavaScript`, natomiast `JSX` jest jego rozszerzeniem, dlatego nie możemy korzystać z propsa o nazwie `class` ponieważ wystąpi błąd w `JSX`. Twórcy Reacta wprowadzili więc props o nazwie `className` do przekazywania odpowiedniej klasy CSS.

```
render() {  
  return <p className="menu navigation-menu">Menu</p>  
}
```

Pytanie 28

Czym są fragmenty?

Jest to wzorzec w Reakcie wykorzystywany do zwrócenia wielu elementów. Fragmenty pozwalają grupować listę komponentów-dzieci bez konieczności wrapeowania dodatkowym elementem (np. elementem `div`).

```
render() {  
  return (  
    <React.Fragment>  
      <ChildComponent />  
      <ChildComponent />  
      <ChildComponent />  
    </React.Fragment>  
  )  
}
```

Pytanie 29

Dlaczego fragmenty są lepsze niż kontenery div?

- Fragmenty są nieco szybsze i używają mniej pamięci, ponieważ nie tworzą dodatkowego węzła w *DOM*. To sprawdza się zdecydowanie przy dużych drzewach
- Kilka rozwiązań takich jak *Flexbox* i *CSS Grid* posiadają specjalne relacje rodzic-dziecko i dodawanie dodatkowych *divów* pomiędzy nimi może spowodować problemy

Pytanie 30

Czym są komponenty bezstanowe?

Jeśli zachowanie komponentu jest niezależne od jego stanu, wówczas może to być komponent bezstanowy. Bardzo często w tym wypadku stosuje się funkcję zamiast klasy do tworzenia tego typu komponentu. Dopóki nie potrzeba korzystać z lifecycle methods, wówczas należy używać komponentów funkcyjnych. To podejście zawiera wiele zalet - komponenty są łatwe do napisania, zrozumienia, testowania i trochę szybsze w działaniu.

Pytanie 31

Czym są komponenty stanowe (stateful components)?

Jeśli zachowanie komponentu zależy od stanu, wtedy komponent jest nazywany komponentem stanowym (*stateful component*). Takie komponenty są komponentami klasowymi i posiadają stan, który jest inicjalizowany w ciele klasy lub konstruktorze.

```
class App extends Component {  
  constructor(props) {  
    super(props)  
    this.state = { count: 0 }  
  }  
  
  render() {  
    // ...  
  }  
}
```

Pytanie 32

Jakie jest zastosowanie metody render w react-dom?

Ta metoda jest używana do renderowania Reactowych elementów w DOMie w określonym kontenerze i zwraca referencję do komponentu. Jeśli element Reactowy był wcześniej wyrenderowany do tego kontenera, to zapewniana jest wtedy aktualizacja DOM.

Pytanie 33

Jak stylować komponenty?

Atrybut style akceptuje obiekt JavaScript pisany camelCasem zamiast standardowo pisanych klas CSS.

```
const style = {
  color: 'blue',
  backgroundColor: 'red'
};

const HelloWorldComponent = () => {
  return <div style={style}>Hello World!</div>
}
```

Pytanie 34

Jak korzystać z dekoratorów?

Możesz wykorzystywać dekoratory w swoich komponentach klasowych, poprzez przekazywanie funkcji do komponentu w następujący sposób:

```
@setTitle('Profile')
class Profile extends React.Component {
    // ...
}

const setTitle = (title) => (WrappedComponent) => {
    return class extends React.Component {
        componentDidMount() {
            document.title = title
        }

        render() {
            return <WrappedComponent {...this.props} />
        }
    }
}
```

Tips: Ten sposób pisania kodu nie jest jeszcze w dniu dzisiejszym zatwierdzony w ES7.

Pytanie 35

Dlaczego należy nazywać komponenty z dużych liter?

Jeśli renderujesz komponent za pomocą JSX, nazwa komponentu powinna zaczynać się z dużej litery w przeciwnym razie React wyrzuci wyjątek związany z nieznanym tagiem elementu. Ta konwencja została wprowadzona, ponieważ tylko elementy *HTML* oraz *SVG* mogą zaczynać się od małych liter.

```
class MyComponent extends React.Component {  
}
```

Pytanie 36

Jak można zmusić komponent do przerenderowania bez korzystania z setState?

Domyślnie, kiedy zmienia się stan komponentu lub komponentu otrzymuje nowe propsy, wówczas automatycznie się przerenderuje. Jeśli metoda render zależy od danych, można wskazać Reactowi, że komponent potrzebuje przerenderowania za pomocą `forceUpdate()`

Jednak zalecane jest aby unikać zastosowania `forceUpdate` i czytać tylko z `this.props` oraz `this.state`

Pytanie 37

Jak zrobić pętlę w JSX?

Można wykorzystywać do tego metodę map razem z funkcjami strzałkowymi. W tym przykładzie tablica obiektów items jest mapowana na tablicę komponentów:

```
<p>
  {items.map(item => <MyComponent key={item.id} name=
  {item.name} />)}
</p>
```

Nie można tego zrobić za pomocą pętli *for*:

```
<p>
  for (let i = 0; i < items.length; i++) {
    <MyComponent
      key={items[i].id}
      name={items[i].name} />
  }
</p>
```

Dzieje się tak ponieważ tagi JSX są transpilowane do wywołań funkcji i nie możesz używać deklaracji wewnętrznych wyrażeń.

Pytanie 38

Jaka jest różnica pomiędzy React a ReactDOM?

Pakiet react zawiera metodę `React.createElement`, `React.Component`, `React.Children` oraz inne pomocnicze funkcje związane z elementami i komponentami. Można o tym myśleć jako „narzędziach” potrzebnych do zbudowania komponentu. Pakiet `react-dom` zawiera `ReactDOM.render()`, a w `react-dom/server` znajdują się helpery związane z renderowaniem po stronie serwera np.: `ReactDOMServer.renderToString()` oraz `ReactDOMServer.renderToStaticMarkup()`

Pytanie 39

Dlaczego ReactDOM jest oddzielony od Reacta?

Zespół Reacta pracował nad wyeksportowaniem wszystkich rzeczy związanych z DOM do oddzielnej biblioteki *ReactDOM*. W React 0.14 znajduje się pierwsze wydanie w której biblioteki są odseparowane. Patrząc na niektóre pakiety takie jak *react-native*, *react-art*, *react-canvas*, oraz *react-three* jasne się stało, że idea Reacta ma już mniej wspólnego z przeglądarkowym DOM.

Aby wykorzystywać Reacta w większej liczbie środowisk rozdzielono biblioteki na dwie paczki *react* oraz *react-dom*. To otwiera drogę do pisania komponentów które mogą być udostępniane pomiędzy środowiskami wykonywania np. React w przeglądarce i React Native w systemach takich jak *iOS* oraz *Android*.

Pytanie 40

Dlaczego nie możesz zmienić wartości propsów?

Filozofia Reacta zakłada, że propsy są niezmienne (*immutable*) i ich przepływ odbywa się z góry na dół. To znaczy, że komponent-rodzic może wysłać propsy tylko do swojego dziecka, ale dzieci nie mogą modyfikować otrzymanych propsów.

Pytanie 41

Jak aktualizować komponent co sekundę?

Należy skorzystać z funkcji `setInterval()` aby by wywołać zmiany, ale należy pamiętać o usunięciu timera po odmontowaniu komponentu, aby zapobiec potencjalnym błędom i wyciekom pamięci.

```
componentDidMount() {  
  this.interval = setInterval(() =>  
    this.setState({ time: Date.now() }),  
    1000  
  )  
}  
  
componentWillUnmount() {  
  clearInterval(this.interval)  
}
```

Pytanie 42

Jak sfocusować element input po załadowaniu strony?

Można to wykonać za pomocą stworzenia referencji do elementu input oraz wykorzystania `componentDidMount()`:

```
class App extends React.Component{  
  
  componentDidMount() {  
    this.nameInput.focus()  
  }  
  
  render() {  
    return (  
      <div>  
        <input defaultValue="Won't focus"/>  
        <input  
          ref={(input) => this.nameInput = input}  
          defaultValue="Will focus"  
        />  
      </div>  
    )  
  }  
}  
  
ReactDOM.render(<App />, document.getElementById('app'))
```

Pytanie 43

Jak zdefiniować stałe?

Można skorzystać ze słowa kluczowego static w ES7:

```
class MyComponent extends React.Component {  
  static DEFAULT_PAGINATION = 10  
}
```

Pytanie 44

Jak wywołać kliknięcie?

Możesz wykorzystać specjalnego propsa ref do przycisku, aby uzyskać referencję do `HTMLInputElement`, przechować tę referencję jako właściwość klasy, a potem użyć jej do późniejszego wywołania event handlerów wykorzystując `HTMLElement.click`. Można to wykonać w dwóch krokach:

- 1) Stwórz ref w metodzie `render`:

```
<input ref={input => this.inputElement = input} />
```

- 2) Wykonaj zdarzenie `click`:

```
this.inputElement.click()
```

Pytanie 45

Czy mogę dispaczować akcję w reducerze?

Dispaczowanie, czyli wykonywanie akcji wewnętrz reducera jest antywzorcem. Reducer powinien nie posiadać efektów ubocznych i zwracać nowy obiekt stanu. Dodawanie listenerów i dispaczowanie akcji wewnętrz reducera może prowadzić do akcji łańcuchowych i innych efektów ubocznych.

Pytanie 46

Czy muszę przechowywać cały stan komponentu w Reduksie?

Nie, jak najbardziej możesz korzystać dalej ze stanu wewnętrznego komponentu, natomiast w storze Reduksa przechowywać dane z jakich korzysta komponent.

Pytanie 47

Jak dispaczować akcję w trakcie ładowania aplikacji?

Można dispaczować akcję w `componentDidMount`, a w metodzie `render` zweryfikować dane.

```
class App extends Component {
  componentDidMount() {
    this.props.fetchData()
  }

  render() {
    return this.props.isLoaded
      ? <div>Loaded</div>
      : <div>Not Loaded</div>
  }
}

const mapStateToProps = (state) => ({
  isLoaded: state.isLoaded
})

const mapDispatchToProps = { fetchData }

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(App)
```

Pytanie 48

Jaka jest różnica pomiędzy komponentem a kontenerem?

Komponent jest klasą lub funkcją, która opisuje część prezentacyjną aplikacji.

W kontekście Reduksa kontener jest umownie nazywany komponentem, który jest połączony ze storem Reduksa. Kontenery subskrybują zmiany stanu Reduksa oraz zajmują się dispatchowaniem akcji. Przeważnie nie renderują elementów DOM, a delegują renderowanie do komponentów-dzieci.

Pytanie 49

W jaki sposób można zapisać mapDispatchToProps()?

Jest kilka sposobów na bindowanie kreatora akcji do metody dispatch w `mapDispatchToProps`. Poniżej znajduje się kilka przykładów:

```
const mapDispatchToProps = (dispatch) => ({
  action: () => dispatch(action())
})

const mapDispatchToProps = (dispatch) => ({
  action: bindActionCreators(action, dispatch)
})

const mapDispatchToProps = { action }
```

Pytanie 50

Po co jest parametr ownProps w mapStateToProps i mapDispatchToProps?

Jeśli określmy parametr ownProps, React Redux przekaże propsy, które przekazaliśmy komponentowi do funkcji connect. Więc jeśli używamy zkonnektowanego komponentu:

```
import ConnectedComponent from  
'./containers/ConnectedComponent'  
  
<ConnectedComponent user="Janusz" />
```

Wartość ownProps wewnętrz mapStateToProps() oraz mapDispatchToProps() będzie obiektem: { user: 'Janusz' }.

Pytanie 51

Jak zaprojektować strukturę aplikacji, która korzysta z Reduksa?

Większość aplikacji ma kilka folderów zawierających:

- *components* - używane do przechowywania „dumb componentów” niezleżnych od Reduksa
- *containers* - używane do „smart components” - komponentów połączonych z Reduksem
- *actions* - używane przez kreatory akcji
- *reducers* - używane przez reducery
- *store* - używane do inicjalizacji storów
- *constants* - przechowywanie stałych, które są używane przez akcje, reducery i czasami middleware (np. *redux-saga*)

Pytanie 52

Czym jest akcja w Reduksie?

Akcje są czystymi obiektami JavaScript przechowującymi informacje, które są wysyłane z aplikacji do store raz są jedynym źródłem danych w store. Akcje muszą posiadać właściwość `type`, która określa typ akcji, jaka zostanie wywołana.

Przykład akcji dodającej nowe zadanie:

```
{  
  type: ADD_TASK,  
  text: 'Add new task'  
}
```

Pytanie 53

Czy Redux może być wykorzystywany tylko z Reactem?

Redux może być wykorzystany jako kontener z danymi dla dowolnej warstwy UI. Najczęściej wykorzystuje się Reduksa w React oraz React Native, ale również można wykorzystywać go w *Angular*, *Angular 2+* czy *Vue*. Redux dostarcza po prostu mechanizm subskrypcji, który może być wykorzystywany przez dowolny kod.

Pytanie 54

Jakie jest zastosowanie registerServiceWorker?

React tworzy service worker domyślnie bez konieczności konfiguracji. Service worker jest usługą, która pozwala na kieszowanie assetów lub plików w przypadku, gdy użytkownik znajdzie się w trybie offline lub korzysta z wolnego połączenia z Internetem. Wówczas będzie mógł cały czas poruszać się po ekranach aplikacji, a aplikacja uzyska lepszy UX.

```
import React from 'react'
import ReactDOM from 'react-dom'

import App from './App'
import registerServiceWorker
  from './registerServiceWorker'

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
registerServiceWorker()
```

Pytanie 55

Czym są Hooki w Reakcie?

Hooki wprowadziły nowe możliwości - korzystanie ze stanu komponentu bez konieczności pisania klasy. Przykład z wykorzystaniem `useState`:

```
import { useState } from 'react'

function Counter() {
  const [count, setCount] = useState(0)

  return (
    <div>
      <p>You've clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  )
}
```

Ten sam komponent zapisany w postaci klasy wyglądałby następująco:

Pytanie 55

Czym są Hooki w Reakcje?

```
class Counter extends React.Component {  
  
  state = {  
    count: 0  
  }  
  
  setCount = (value) => {  
    this.setState({  
      counter: value + 1  
    })  
  }  
  
  render() {  
    const { count } = this.state;  
    return (  
      <div>  
        <p>You've clicked {count} times</p>  
        <button onClick={() => this.setCount(count + 1)}>  
          Click me  
        </button>  
      </div>  
    )  
  }  
}
```

Pytanie 56

Jakie są zasady stosowania Hooków?

Należy kierować się dwiema zasadami:

- 1) Wywoływać hooki tylko na początku funkcji – nie wolno tego robić wewnętrz pętli, instrukcji warunkowych czy zagnieżdżonych funkcji. To gwarantuje, że hooki będą wywoływane zawsze w tej samej kolejności pomiędzy różnymi komponentami.
- 2) Wywoływać hooki tylko z funkcji, które zawierają instrukcje Reacta – nie powinno się ich wykonywać w czystym JavaScript.

Pytanie 57

Czym jest funkcja lazy?

Funkcja React.lazy pozwala renderować dynamicznie importowane komponenty jak zwykłe komponenty. Funkcja jako argument przyjmuje funkcję, która wywołuje dynamiczny import. Ta musi zwrócić obiekt typu Promise, który zostanie rozwiązany jako moduł z domyślnym eksportem zawierającym komponent Reacta.

Przykład:

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <OtherComponent />
    </div>
  )
}
```

Bonusy

Ufff udało Ci się dotrwać do końca :) Mam nadzieję, że zdobytą wiedza pomoże Ci sprawnie przejść proces rekrutacji.

W ramach nagrody przygotowałem dla Ciebie kilka bonusów, jest to zestaw bibliotek z którymi warto się zapoznać, gdy zaczniesz pisać bardziej zaawansowane aplikacje. Ja korzystam z nich na codzień i bardzo ułatwiają życie:

- React Redux - biblioteka ułatwiająca korzystanie z Reduksa w React
- React Router - router dla Reacta
- Redux Saga - middleware obsługujący akcje asynchroniczne
- Jest - biblioteka do testowania aplikacji i komponentów
- Styled-components - biblioteka do stylowania komponentów
- Material UI - zestaw gotowych komponentów

Kod zniżkowy

Od jakiegoś czasu pracuję nad merytorycznym kursem Reacta w formie online, a ponieważ ta wiedza może Ci się przydać, pomyślałem że zainteresuje Cię kod zniżkowy :) W momencie uruchomienia sprzedaży (sprzedaż potrwa tylko 8 dni) możesz wykorzystać kod zniżkowy **REACT20** obniżający wartość zamówienia o **10%**.

Do usłyszenia już niebawem
Patryk Omotek