



**POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI**

**KIERUNEK STUDIÓW
INFORMATYKA**

***MATERIAŁY DO ZAJĘĆ
LABORATORYJNYCH***

Szkielety programistyczne w aplikacjach internetowych

dr Mariusz Dzieńkowski

Lublin 2022

LABORATORIUM 6B. BUDOWA APLIKACJI WARSTWY USŁUG APLIKACJI INTERNETOWYCH NA PODSTAWIE BIBLIOTEKI REACT

Cel laboratorium:

Nabycie umiejętności posługiwania się biblioteką React podczas implementacji aplikacji funkcjonującej po stronie klienta.

Zakres tematyczny zajęć:

- Tworzenie i uruchamianie aplikacji opartej na bibliotece React.
- Implementacja komponentów w React za pomocą funkcji oraz klas.
- Sposoby przekazywania i odbierania danych przez komponenty (*props* i *state*).
- Komponenty bezstanowe i ze stanem.
- Obsługa formularzy przy pomocy biblioteki React.

Pytania kontrolne:

- a) Czym jest i do czego służy JSX (JavaScript XML)?
- b) Przedstaw strukturę aplikacji opartej na bibliotece React.
- c) Opisz strukturę komponentu zbudowanego na bazie funkcji oraz klasy.
- d) Czym są i do czego służą właściwości (*props*) i stany (*state*)?

Zadanie 6.4. Aplikacja „Lista zakupów”

Zaimplementuj aplikację „Lista zakupów” wykorzystując do tego celu bibliotekę React. Na rysunku 6.5 zaprezentowano widok interfejsu aplikacji z zaznaczonymi komponentami.

The screenshot shows a web application titled "Lista zakupów" (Shopping List). It features a form with three input fields: "Produkt" (Product) with a placeholder "Nazwa produktu", "Kategoria" (Category) with a dropdown menu showing "Owoce i warzywa", and "Ilość" (Quantity) with a value of "1". Below the form is a "Dodaj" (Add) button. Underneath the form is a list of items already added to the shopping list, each in a separate row with a delete button (X):

- mleko (milk) with quantity 1
- chleb (bread) with quantity 1
- jablka (apples) with quantity 2

Red rectangular boxes are drawn over the interface to highlight specific components: one box encloses the title "Lista zakupów"; another box encloses the entire form area (inputs and buttons); a third box encloses the list of items; and a fourth box encloses the "chleb" item row.

Rys. 6.5 Aplikacja Lista zakupów z zaznaczonymi komponentami

Etap 1. Utworzenie i uruchomienie aplikacji React

```
npx create-react-app moja-aplikacja  
npm start
```

Efekt działania aplikacji będzie widoczny w przeglądarce po przejściu pod adres `localhost:3000`.

Etap 2. Tworzenie komponentów

Komponenty definiujemy za pomocą funkcji, która zwraca element Reacta. W folderze `src` należy utworzyć nowy folder o nazwie `components` oraz plik `Header.js` (Listing 6.9). W nim będzie zdefiniowany pierwszy komponent.

Listing 6.9 Kod komponentu `Header` w postaci funkcji

```
const Header = () => {  
  return (  
    <h2>Lista zakupów</h2>  
  )  
}  
export default Header
```

Jest to najprostszy sposób na stworzenie komponentu, jednak możliwe jest również użycie w tym celu klasy tak jak na poniższym przykładzie (Listing 6.10).

Listing 6.10 Kod komponentu `Header` w postaci klasy

```
class Header extends React.Component {  
  render() {  
    return (  
      <h2>Lista zakupów</h2>  
    )  
  }  
}  
export default Header
```

Elementy tworzymy przy użyciu składni JSX. Przypomina ona HTML, ale umożliwia wstawianie znaczników i dodatkowo daje pełnię możliwości języka JavaScript. Atrybut `class` w JSX zmienia się w `className`.

W celu użycia komponentu w innym komponencie należy go najpierw zaimportować. Wówczas komponent `Header` będzie należał do głównego komponentu `App` (Listing 6.11).

Listing 6.11 Import komponentu `Header` do głównego komponentu `App`

```
import './App.css';  
import Header from './components/Header'  
function App() {  
  return (  
    <div className="App">  
      <Header />  
    </div>  
  )  
}
```



```
)  
}  
  
export default App
```

Dane do komponentu można przekazać za pomocą argumentu *props* (oznaczającym właściwości), który jest obiektem przenoszącym dane. Poniżej przedstawiony jest przykład przekazania tytułu nagłówka (Listing 6.12 i 6.13).

Listing 6.12 Przekazywanie danych komponentu za pomocą argumentu *props*

```
const Header = (props) => {  
  return (  
    <div className='header'>  
      <h2>{props.title}</h2>  
    </div>  
  )  
}  
  
export default Header
```

Listing 6.13 Kod komponentu głównego *App*

```
import './App.css';  
import Header from './components/Header'  
  
const App = () => {  
  return (  
    <div className="App">  
      <Header title='Lista zakupów'>  
    </div>  
  )  
}  
  
export default App
```

Do zmiennych odwołujemy się, otaczając je nawiasami klamrowymi. Możemy ustawić domyślną wartość, która zostanie wyświetlona, gdy nie zostanie przekazany żaden argument oraz ustawić typ danej właściwości za pomocą biblioteki *prop-types*. Zamiast używać obiektu *props* możemy również przekazać argument w sposób przedstawiony poniżej (Listing 6.14).

Listing 6.14 Odwoływanie się do zmiennych

```
const Header = ({title}) => {  
  return <h2>{title}</h2>  
}  
  
export default Header
```

Etap 3. Stan komponentów



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska

Unia Europejska
Europejski Fundusz Społeczny



Stan (ang. state) służy do przechowywania informacji należących do komponentu. Gdy stan ulegnie zmianie, komponent zostanie ponownie zrenderowany. Do definiowania stanu warto używać tzw. hooków. Jest to nowy dodatek w React, który pozwala używać stanu i innych funkcjonalności w tej bibliotece, bez użycia klas.

Oto kod tworzący komponent Products (Listing 6.15).

Listing 6.15 Kod komponentu Products.js

```
export const products_list = [
  {
    "id": 1,
    "name": "mleko",
    "category": "diary",
    "quantity": 1
  },
  {
    "id": 2,
    "name": "chleb",
    "category": "bread",
    "quantity": 1
  },
  {
    "id": 3,
    "name": "jabłko",
    "category": "fruit&vagetables",
    "quantity": 2
  }
]
const Products = ({products}) => {
  return (
    <div>
      {products.map((product) => <h3 key={product.id}>{product.name}</h3>)}
    </div>
  )
}
export default Products
```

Po tej modyfikacji Kod głównego komponentu aplikacji – plik App.js (Listing 6.16).

Listing 6.16 Kod główneog komponentu po dołączeniu do niego komponentu Products

```
import { useState } from "react"
import Header from './components/Header'
import Products, { products_list } from './components/Products'
import './App.css';

function App() {
  const [products, setProducts] = useState(products_list)
  return (
    <div className="App">
      <Header title="Lista zakupów" />
    </div>
  )
}
```

```

    <Products products={products} />
  </div>
)
}

export default App

```

Wcześniej dane zostały wczytane z tablicy, teraz będą przeniesione do stanu wewnątrz funkcji komponentu. Najpierw należy zacząć od importu hooka *useState*. Następnie za pomocą hooka deklarujemy zmienną stanu *products* przechowującą zadania. Jako argument przyjmujemy wartość początkową stanu, czyli dane o naszych zadaniach. Oprócz deklaracji zmiennej stanu deklarowana jest również funkcja *setProducts* pozwalająca na aktualizowanie stanu.

Listing 6.17 Przeniesienie danych do stanu wewnątrz funkcji komponentu (plik *Products.js*)

```

import { useState } from "react"

const Products = () => {
  const [products, setProducts] = useState([
    {
      "id": 1,
      "name": "mleko",
      "category": "diary",
      "quantity": 1
    },
    {
      "id": 2,
      "name": "chleb",
      "category": "bread",
      "quantity": 1
    },
    {
      "id": 3,
      "name": "jabłka",
      "category": "fruit&vagetables",
      "quantity": 2
    }
  ])
  return (
    <div>
      {products.map((product) => <h3 key={product.id}>{product.name}</h3>)}
    </div>
  )
}

export default Products

```

Listing 6.18 Główny komponent po modyfikacji

```

import Header from './components/Header'
import Products from './components/Products'
import './App.css'

```

```
function App() {  
  return (  
    <div className="App">  
      <Header title="Lista zakupów" />  
      <Products />  
    </div>  
  )  
}  
  
export default App
```

Po to, aby móc korzystać ze stanu we wszystkich komponentach powinien on się znajdować w komponencie nadrzędnym. Dlatego należy przenieść utworzony stan do komponentu *App* i przekazać go za pomocą *props* do komponentu *Products* (Listing 6.19).

Listing 6.19 Przeniesienie utworzonego stanu do komponentu *App*

```
import { useState } from "react"  
import Header from './components/Header'  
import Products from './components/Products'  
import './App.css'  
  
function App() {  
  const [products, setProducts] = useState(  
    [  
      {  
        "id": 1,  
        "name": "mleko",  
        "category": "diary",  
        "quantity": 1  
      },  
      {  
        "id": 2,  
        "name": "chleb",  
        "category": "bread",  
        "quantity": 1  
      },  
      {  
        "id": 3,  
        "name": "jabłko",  
        "category": "fruit&vegetables",  
        "quantity": 2  
      }  
    ]  
  )  
  return (  
    <div className="App">  
      <Header title="Lista zakupów" />  
      <Products />  
    </div>
```

```
)
}
export default App
```

W następnym etapie należy utworzyć komponent *Product*, który będzie reprezentował pojedynczy produkt. Dodatkowo, aby użyć ikon należy zainstalować bibliotekę *react-icons* za pomocą npm oraz zaimportować ją do komponentu. W tym momencie należy plik *App.css* zastąpić nowymi regułami, aby uzyskać efekt podobny do przedstawionego na Rys. 6.6.



Rys. 6.6 Aplikacja z komponentami *Product*

Listing 6.20 Implementacja komponentu *Product.js*

```
const Product = ({ product }) => {
  return (
    <div className='product'>
      <div className='product-info'>
        <div className='input-name'>
          <input type='checkbox' value='' required />
          <p>{product.name}</p>
        </div>
        <p>{product.quantity}</p>
      </div>
      <div className='product-icons'>
      </div>
    </div>
  )
}

export default Product
```

Listing 6.21 Modyfikacja pliku *Products.js*

```
import Product from './Product'

const Products = ({products}) => {
  return (
    <div className='products'>
      {products.map((product) => (
        <Product key={product.id} product={product} />
      ))}
    </div>
  )
}
```



```
)  
}  
export default Products
```

W głównym komponencie należy tylko zmodyfikować wywołanie komponentu Products:
<Products products={products}/>

Listing 6.22 Reguły stylistyczne zawarte w pliku App.css

```
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400&display=swap');  
body{  
  display: flex;  
  justify-content: center;  
  background-color: #2f4f4f;  
  font-family: 'Poppins', sans-serif;  
}  
.App {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  justify-content: center;  
  background-color: white;  
  width: 350px;  
  padding: 20px 0;  
  margin: 20px 0;  
  border-radius: 6px;  
}  
.products{  
  width: 300px;  
}  
.product{  
  display: flex;  
  align-items: center;  
  background-color: #CEB5A7;  
  padding: 0 15px;  
  margin-bottom: 10px;  
  color: white;  
  border-radius: 6px;  
}  
.product-info{  
  display: flex;  
  justify-content: space-between;  
  flex: 0.7;  
}  
.checked{  
  text-decoration: line-through;  
}  
.product-info .input-name{  
  display: flex;  
}
```



```
.product-info .input-name p{
  margin-left: 10px;
}
.product-icons{
  display: flex;
  justify-content: flex-end;
  align-self: center;
  flex: 0.3;
}
```

Etap 4. Modyfikacja stanu

Aktualnie stan znajduje się w głównym komponencie *App*, dlatego wszystkie metody go modyfikujące również powinny znajdować się w tym miejscu, wewnątrz funkcji *App.js*, a przed słowem kluczowym *return*. Usuwanie zadania z listy można zrealizować w kilku krokach.

Listing 6.23 Usuwanie zadania z listy

```
const deleteProduct = (id) => {
  setProducts(products.filter((product) => product.id !== id))
}
```

Za pomocą metody *setProducts* modyfikujemy początkową wartość stanu, nadpisując ją nową tablicą zadań z wykluczeniem usuwanego produktu. W tym momencie należy także zmodyfikować wywołanie komponentu *Products* (Listing 6.24).

Listing 6.24 Wywołanie komponentu *Products*

```
<Products products={products} onDelete={deleteProduct}/>
```

Metodę *deleteProduct* przekazujemy za pomocą właściwości do komponentu *Product* i następnie przypisujemy do zdarzenia *onClick* w elemencie ikony usuwania (Listing 6.25). Wcześniej należy zainstalować ikony Reacta: **npm i react-icons --save**, a następnie je zaimportować w pliku *Product.js*: **import { FiX } from "react-icons/fi"**

Listing 6.25 Wywołanie metody *onClick* jako zdarzenie kliknięcia na ikonie usuwania *Fix* (komponent *Product*)

```
<FiX onClick={() => onDelete(product.id)}></FiX>
```

Składnia JSX oprócz znaczników pozwala na używanie możliwości JavaScript. Zostanie to zastosowane do wyświetlenia komunikatu, gdy na liście nie ma żadnych zadań, za pomocą prostej instrukcji warunkowej.

Listing 6.26 Wyświetlanie komunikatu, gdy na liście nie ma żadnych produktów (*App.js*)

```
{products.length > 0 ? <Products products={products} onDelete={deleteProduct}/>:<p>Brak
produktów</p>}
```

Wyrażenie warunkowe należy także wykorzystać, aby po zaznaczeniu pola checkbox uzyskać efekt przekreślenia nazwy produktu, tak jak to pokazuje rysunek 6.7.



Rys. 6.7 Aplikacja Lista zakupów – efekt przekreślenia produktu

Na koniec tego etapu należy jeszcze zmodyfikować kod pliku Products.js

Listing 6.27 Modyfikacja komponentu Products

```
import Product from './Product'

const Products = ({products, onDelete}) => {
  return (
    <div className='products'>
      {products.map((product) => (<Product key={product.id} product={product} onDelete={onDelete}
/>))}
    </div>
  )
}
export default Products
```

Listing 6.28 Kod komponentu Product w rezultacie ostatnich działań

```
import { useState } from "react"
import { FiX } from "react-icons/fi"

const Product = ({ product, onDelete }) => {
  const [isChecked, setIsChecked] = useState(false)
  return (
    <div className='product'>
      <div className='product-info'>
        <div className='input-name'>
          <input type='checkbox' value={isChecked}
            onChange={(e) => setIsChecked(!isChecked)}
            required/>
          <p className={
            isChecked ? 'checked' : ''}>{product.name}</p>
        </div>
        <p>{product.quantity}</p>
      </div>
      <div className='product-icons'>
        <FiX onClick={() => onDelete(product.id)}></FiX>
      </div>
    </div>
  )
}
export default Product
```

Po zastosowaniu modyfikacji w bieżącym etapie wygląd aplikacji będzie taki jak na rysunku 6.8.

Rys. 6.8 Widok formularza z zaimplementowaną funkcjonalnością usuwania produktów

Etap 5. Dodawanie nowych zakupów do listy

Utwórz komponent *AddProduct*, tak aby wyglądał podobnie do przedstawionego poniżej i zaimportuj go w pliku *App.js*.

Rys. 6.9 Okno dodawania nowego produktu

Do dodania nowego produktu potrzebne będą informacje wprowadzone poprzez formularz. Do monitorowania i zapisu tych danych można użyć stanów na poziomie komponentu.

Listing 6.29 Przechwycenie danych z formularza

```
const [name, setName] = useState('')
const [category, setCategory] = useState('fruit&vagetables')
const [quantity, setQuantity] = useState(1)
```

Wprowadzenie tekstu w polu input lub zaznaczenie pola checkbox będzie wywoływało modyfikację stanu. Można tego dokonać przy pomocy metody *onChange*.

Listing 6.30 Modyfikacja stanu za pomocą metody *onChange*

```
<input type='text' placeholder='Nazwa produktu' value={name} onChange={(e) =>
setName(e.target.value)} required/>
```

Do modyfikacji stanu globalnego należy zdefiniować, a następnie wywołać metodę *addProduct* z głównego komponentu i przekazać niezbędne dane w metodzie *onSubmit* z formularza.

Listing 6.31 Definicja metody *addProduct()* i przekazanie do niej danych z formularza

```
const addProduct = (product) => {
  const id = products.length + 1
  const newProduct = { id, ...product }
  setProducts([...products, newProduct])
}
```

```
}
```

```
...
```

```
<AddProduct onAdd={addProduct} />
```

Cały kod komponentu AddProduct zawiera Listing 6.32.

Listing 6.32 Kod komponentu AddProduct

```
import { useState } from "react"

const AddProduct = ({ onAdd }) => {
  const [name, setName] = useState("")
  const [category, setCategory] = useState('fruit&vagetables')
  const [quantity, setQuantity] = useState(1)
  const add = (e) => {
    e.preventDefault()

    onAdd({ name, category, quantity })
    setName("")
    setCategory('fruit&vagetables')
    setQuantity(1)
  }
  return (
    <form className='add-list' onSubmit={add}>
      <div className='input-div'>
        <label>Produkt</label>
        <input type='text' placeholder='Nazwa produktu' value={name} onChange={(e) =>
setName(e.target.value)} required />
      </div>
      <div className='input-div'>
        <label>Kategoria</label>
        <select value={category} onChange={(e) => setCategory(e.target.value)}>
          <option value='fruits&vagetables'>Owoce i warzywa</option>
          <option value='dairy'>Nabiał</option>
          <option value='bread'>Pieczywo</option>
        </select>
      </div>
      <div className='input-div'>
        <label>Ilość</label>
        <input type='number' placeholder='0' value={quantity} min="0" onChange={(e) =>
setQuantity(e.target.value)} />
      </div>
      <input type='submit' className='btn' value='Dodaj' />
    </form>
  )
}

export default AddProduct
```

W pliku głównego komponentu należy wykonać 4 czynności:



Fundusze Europejskie
Wiedza Edukacja Rozwój



**Rzeczpospolita
Polska**

Unia Europejska
Europejski Fundusz Społeczny



- zaimportować kod komponentu addProduct

Listing 6.33 Import komponentu addProduct

```
import AddProduct from './components/AddProduct'
```

- dodać w ciele komponentu kod metody dodającej nowe produkty:

Listing 6.34 Kod metody addProduct

```
const addProduct = (product) => {  
  const id = products.length + 1  
  const newProduct = { id, ...product }  
  setProducts([...products, newProduct])  
}
```

- wywołać nowy komponent umieszczając go w sekcji return, w odpowiedniej kolejności

Listing 6.35 Wywołanie komponentu AddProduct

```
<AddProduct onAdd={addProduct} />
```

- zabezpieczyć sytuację braku produktu poprzez wyświetlenie odpowiedniego komunikatu

Listing 6.36 Wygenerowanie komunikatu

```
{products.length > 0 ? <Products products={products} onDelete={deleteProduct} /> : <p>Brak  
produktów</p>}
```

Kolejną czynnością jest dodanie stylów dla utworzonego formularza. Poniższy kod należy dodać do pliku App.css.

Listing 6.37 Arkusz stylów App.css

```
.add-list{  
  display: flex;  
  flex-direction: column;  
  width: 300px;  
  margin-bottom: 20px;  
}  
.input-div{  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  margin-bottom: 10px;  
  max-height: 30px;  
}  
.input-div label{  
  margin-right: 10px;  
}  
.input-div select, input{  
  padding: 5px;  
  background-color: white;  
  border: 1px solid gray;  
  border-radius: 6px;
```



```
}  
.btn{  
  color: white;  
  background-color: #A17C6B;  
  border: transparent;  
  border-radius: 3px;  
  padding: 5px 0;  
  border-radius: 6px;  
}  
input[type=number] {  
  -moz-appearance: textfield;  
}
```

Efekt ostatnich działań będzie formularz pokazany na rysunku 6.10.

The image displays two side-by-side screenshots of a web application titled 'Lista zakupów' (Shopping List).
The left screenshot shows the form with the following state:
- 'Produkt' field: 'Nazwa produktu'
- 'Kategoria' dropdown: 'Owoce i warzywa'
- 'Ilość' field: '1'
- 'Dodaj' button: visible
- List of items:
 - mleko (1) with a delete icon (x)
 - chleb (1) with a delete icon (x)
 - jabłka (2) with a delete icon (x)
 - masło (3) with a delete icon (x)
The right screenshot shows the same form but with the 'Brak produktów' (No products) message at the bottom, indicating the list is empty.

Rys. 6.10 Aplikacja Lista zakupów – efekt dodawania nowych produktu

Etap 6. Wyświetlanie ikon obrazujących dodawane kategorie produktów

Do komponentu Product należy:

- zaimportować 3 ikony

Listing 6.38 Import ikon

```
import { GiMilkCarton, GiSlicedBread, GiShinyApple } from 'react-icons/gi'
```

- dodać w bloku `<div className='product'>` poniższy kod

Listing 6.39 Dołączenie ikon

```
{product.category === "diary" && <GiMilkCarton className='category'></GiMilkCarton> }
```

- dodać wg powyższego wzoru ikony dla pozostałych dwóch kategorii