Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

KIERUNEK STUDIÓW
INFORMATYKA

MATERIAŁY DO ZAJĘĆ LABORATORYJNYCH

Szkielety programistyczne w aplikacjach internetowych

dr Mariusz Dzieńkowski

Lublin 2022

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

LABORATORIUM 6. 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 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)*?

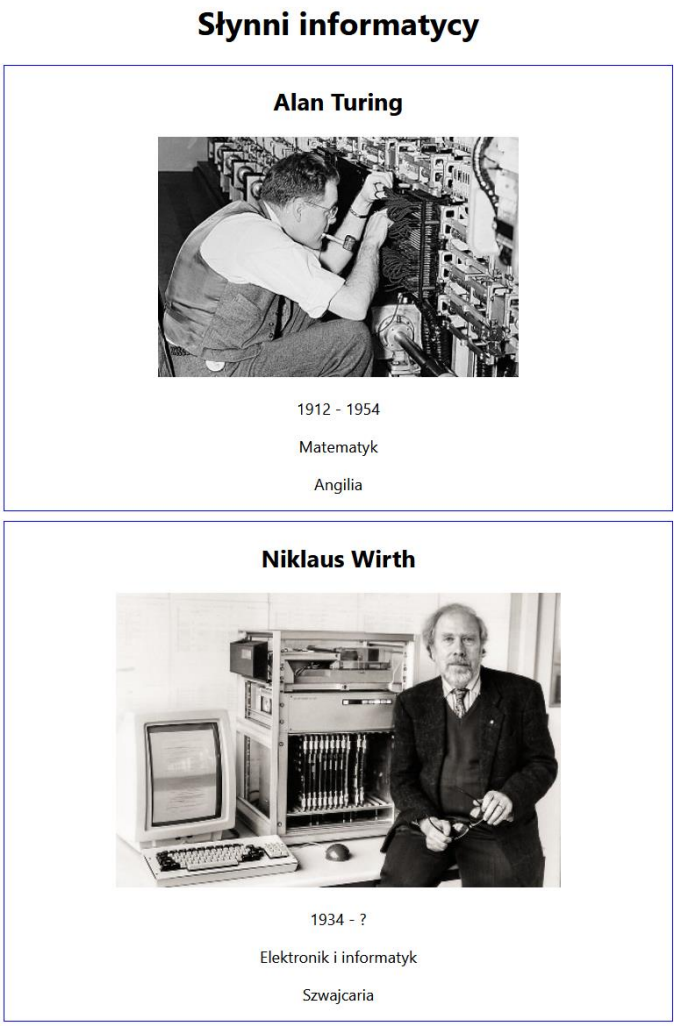
Zadanie 6.1. Aplikacja wyświetlająca karty z informacami o sławnych informatykach

Zrzut ekranowy przedstawiony na rysunku 6.1 to widok aplikacji przedstawiającej dwie ważne postaci informatyki. Aplikacja składa się ze zmodyfikowanego komponentu głównego App, który zawiera powtarzalny kod (Listing 6.1), wyświetlający krótkie informacje o sławnych informatykach. Dostęp do zdjęć: *alan_turing.jpg*, *nicolas_wirth.jpg*, *dennis_ritchie.jpg* i *bjarne_stroustrup.jpg* jest możliwy za pomocą adresu: https://mdz.cs.pollub.pl/ai/ z dodaną odpowiednią nazwą pliku. Natomiast ostylowanie kart przedstawia Listing 6.1.

Listing 6.1 *Plik App.js - kod głównego komponentu aplikacji*

```
import './Card.css'

function App() {
 return (
  <div>
   <h1>Słynni informatycy</h1>
   <div className="Card">
    <h2>Alan Turing</h2>
    <img src="https://mdz.cs.pollub.pl/ai/alan_turing.jpg" alt="Alan Turing" />
    <p>1912 - 1954</p>
    <p>Matematyk</p>
    <p>Angilia</p>
   </div>
   <div className="Card">
    <h2>Niklaus Wirth</h2>
```

```
      <img src="https://mdz.cs.pollub.pl/ai/nicolas_wirth.jpg" alt="Niklaus Wirth" />
      <p>1934 - ?</p>
      <p>Elektronik i informatyk</p>
      <p>Szwajcaria</p>
    </div>
    <div className="Card">
      <h2>Dennis Ritchie</h2>
      <img src="https://mdz.cs.pollub.pl/ai/dennis_ritchie.jpg" alt="Dennis Ritchie" />
      <p>1941 - 2011</p>
      <p>Matematyk, fizyk, informatyk</p>
      <p>USA</p>
    </div>
  </div>
 )
}

export default App
```



Rys. 6.1 Wygląd interfejsu aplikacji zawierającej 2 karty

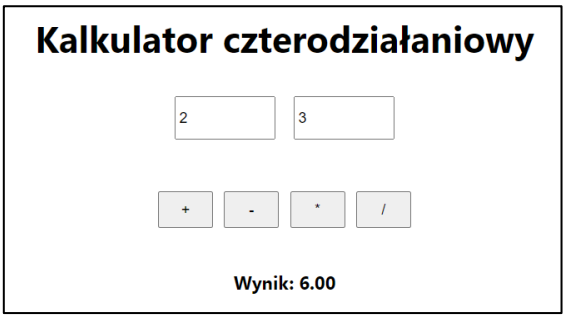
Listing 6.2 *Plik Card.css – zastosowane w aplikacji style*

```
h1 {
  text-align: center;
}
.Card {
  margin-left: auto;
  margin-right: auto;
  margin-top: 10px;
  text-align: center;
  border: 1px solid blue;
  width: 30%;
  display: block;
}
```

- Dodać do aplikacji jeszcze dwie karty zawierające informacje o dwóch innych znanych informatykach (np. Bjarne Stroustrup i Dennis Ritche).
- Na podstawie kodu zawartego w komponencie App.js utworzyć 4 komponenty funkcyjne o nazwach *Card1*, *Card2*, *Card3*, *Card4* i umieścić ich kod w osobnych plikach. W komponencie głównym oprócz nagłówka Słynni informatycy, wywołaj wszystkie utworzone komponenty.
- Zmodyfikować aplikację w ten sposób, aby używała definicji jednego, uniwersalnego komponentu *Card*, do którego dane byłyby przekazywane przez parametr funkcji (tzw. props).

Zadanie 6.2. Kalkulator czterodziałaniowy

Aplikacja z rys. 6.2 zawiera cały kod w komponencie głównym App.



Rys. 6.2 Wygląd aplikacji przed modyfikacją

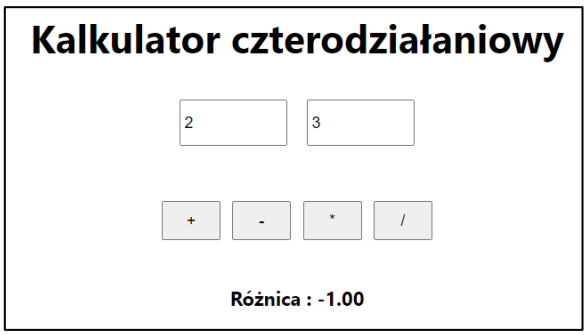
Poniżej znajduje się kod tej aplikacji, który należy najpierw uruchomić, a następnie przekształcić (uzupełnić) w ten sposób, aby aplikacja zamiast słowa *Wynik* wyświetlała nazwę operacji tzn. *Suma*, *Różnica*, *Iloczyn* oraz *Iloraz*. Docelowy wynik działania aplikacji przedstawia rys. 6.3.

Listing 6.3 *Plik głównego komponentu App.js*

```
import { useState } from "react"
import './App.css'
```

```
function App() {
  const [result, setResult] = useState(null)
  const [input1, setInput1] = useState(null)
  const [input2, setInput2] = useState(null)

  const calculate = (e) => {
    let res = eval(`${input1} ${e.target.innerHTML} ${input2}`).toFixed(2)
    setResult(res)
  }

  const firstInput = (e) => {
    let value1 = e.target.value
    setInput1(value1)
  }

  const secondInput = (e) => {
    let value2 = e.target.value
    setInput2(value2)
  }

  return (
    <div className="App">
      <h1>Kalkulator czterodziałaniowy</h1>
      <div>
        <span>
          <input
            type="number"
            onChange={firstInput}
            style={{ width: "5rem", height: "2rem", margin: "0.5rem" }}
          />
        </span>
        <span>
          <input
            type="number"
            onChange={secondInput}
            style={{ width: "5rem", height: "2rem", margin: "0.5rem" }}
          />
        </span>
      </div>
      <div style={{ margin: "2rem" }}>
        <button onClick={calculate} style={{ margin: "0.3rem", width: "3rem", height: "2rem" }}>
          +
        </button>
        <button onClick={calculate} style={{ margin: "0.3rem", width: "3rem", height: "2rem" }}>
          -
        </button>
        <button onClick={calculate} style={{ margin: "0.3rem", width: "3rem", height: "2rem" }}>
          *
```

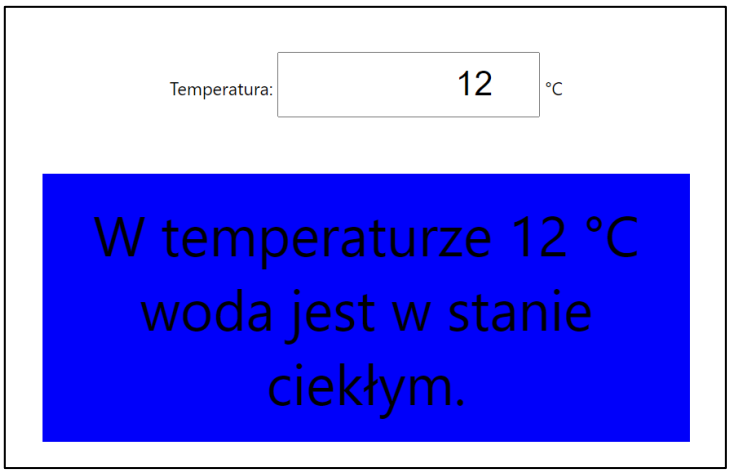
```
      </button>
      <button onClick={calculate} style={{ margin: "0.3rem", width: "3rem", height: "2rem" }}>
        /
      </button>
    </div>
    <h4>Wynik: {result}</h4>
  </div>
 )
}
export default App
```



Rys. 6.3 Wygląd aplikacji po modyfikacji

Zadanie 6.3. Aplikacja określająca stan materii

Aplikacja z rys. 6.4 służy do określania stanu materii wody w zależności od wprowadzonej wartości temperatury w stopniach Celsjusza. Cały kod aplikacji znajduje się w komponencie głównym *App*. Zastosowano w nim tzw. hooka *useEffect*, którego użycie jest informacją dla Reacta, że komponent *App* musi wykonać pewne czynności po jego wyrenderowaniu. Lokalizacja tego hooka wewnątrz komponentu umożliwia dostęp do zmiennej stanu.



Rys. 6.4 Wygląd działającej aplikacji Stan materii

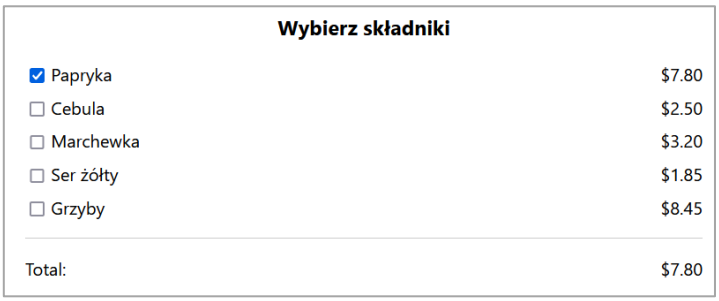
***Listing 6.4** Plik głównego komponentu App.js*

```
import { useState, useEffect } from 'react'
import "./App.css"

function App() {
 const [temperature, setTemperature] = useState(0)
 const [stateMatter, setStateMatter] = useState("")

 const handleChange = (event) => {
  setTemperature(event.target.value)
 }

 useEffect(() => {
  if (temperature <= 0) {
   setStateMatter("stały")
  } else if (temperature >= 100) {
   setStateMatter("gazowy")
  } else {
   setStateMatter("ciekły")
  }
 }, [temperature])

 return (
  <div className="temperature">
   <label>Temperatura: 
    <input
     type="text"
     onChange={handleChange}
     value={temperature}
     placeholder="Wprowadź temperaturę wody"
    /> °C
   </label>
   <div className={stateMatter}>
    <p>
     W temperaturze {temperature} °C woda jest w stanie
     <span > {stateMatter}m.</span>
    </p>
   </div>
  </div>
 )
}
export default App
```

Należy ostylować aplikację, aby zmieniała kolor paragrafu (znacznik <p>) w zależności od stanu skupienia wody (stan stały: kolor czarny, stan ciekły: niebieski, stan lotny: jasnoszary).

Zadanie 6.4. Aplikacja *Lista składników*

Aplikacja z rys. 6.5 będzie zawierała listę produktów, ich cenę oraz podsumowanie. Będą one się znajdowały w pliku toppings.js w postaci listy obiektów. Interfejs aplikacji zawiera głównie komponenty *checkbox*.



Rys. 6.5 Wygląd działającej aplikacji Stan materii

Listing 6.5 *Plik toppings.js zawierający listę produktów (lista obiektów)*

```
export const toppings = [
  {
    name: "Papryka",
    price: 7.80
  },
  {
    name: "Cebula",
    price: 2.50
  },
  {
    name: "Marchewka",
    price: 3.20
  },
  {
    name: "Ser żółty",
    price: 1.85
  },
  {
    name: "Grzyby",
    price: 8.45
  }
]
```

Listing 6.6 *Plik komponentu głównego App.js*

```
import {useState} from 'react'
import './App.css'
import {toppings} from "./toppings"

const getFormattedPrice = (price) => `$${price.toFixed(2)}`
```

```
function App() {
 const [checkedState, setCheckedState] = useState(
  new Array(toppings.length).fill(false)
 )

 const [total, setTotal] = useState(0)

 const handleOnChange = (position) => {
  const updatedCheckedState = checkedState.map((item, index) =>
  index === position ? !item : item)

 setCheckedState(updatedCheckedState)

 const totalPrice = updatedCheckedState.reduce(
  (sum, currentState, index) => {
   if(currentState == true) {
    return sum + toppings[index].price
   }
   return sum
  },
  0
 )

 setTotal(totalPrice)
 }
 return (
  <div className="App">
   <h3>Wybierz składniki</h3>
   <ul className='toppings-list'>
    {toppings.map(({ name, price}, index) => {
     return (
      <li key={index}>
       <div className='toppings-list-item'>
        <div className='left-section'>
         <input
          type="checkbox"
          id={`custom-checkbox-${index}`}
          name={name}
          value={name}
          checked={checkedState[index]}
          onChange={() => handleOnChange(index)}
          />
          <label htmlFor={`custom-checkbox-${index}`}>{name}</label>
        </div>
        <div className='right-section'>{getFormattedPrice(price)}</div>
       </div>
      </li>
     )
    })}
```

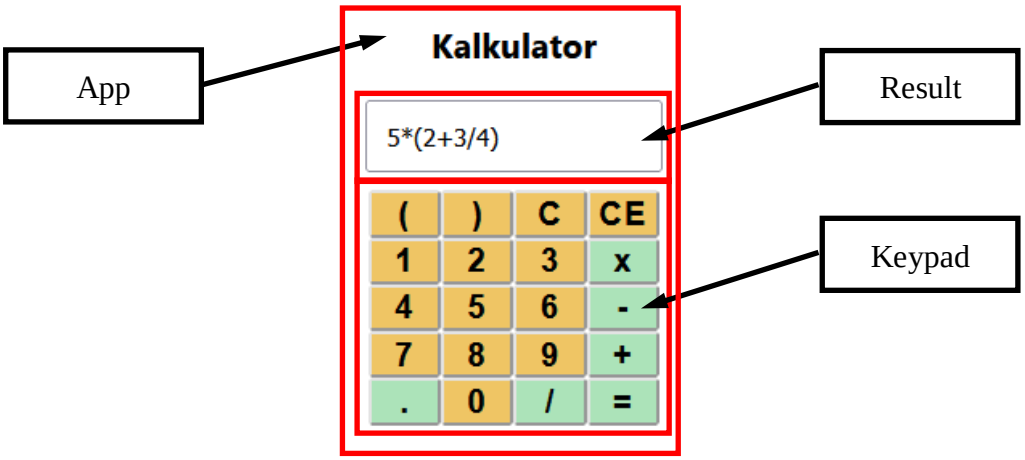
```
      <li>
        <div className='toppings-list-item'>
          <div className='left-section'>Total:</div>
          <div className='right-section'>{getFormattedPrice(total)}</div>
        </div>
      </li>
    </ul>
  </div>
 )
}

export default App
```

***Listing 6.7** Plik App.css zawierający reguły stylistyczne*

```
h3 {
  text-align: center;
}
.topping {
  margin-top: 0.3rem;
  vertical-align: text-bottom;
}
.toppings-list,
.total {
  width: 30%;
  margin: 0 auto;
}
.toppings-list {
  list-style: none;
  padding: 0;
}
.toppings-list li {
  margin-bottom: 0.5rem;
}
.toppings-list-item {
  display: flex;
  justify-content: space-between;
}
.toppings-list li:last-child {
  border-top: 1px solid #ccc;
  margin-top: 1rem;
  padding-top: 1rem;
}
.toppings-list-item label {
  vertical-align: text-bottom;
  margin-left: 0.2rem;
}
.total {
  margin-top: 1rem;
}
```

Zadanie 6.5. Aplikacja *Kalkulator*

Aplikacja będzie potrzebowała dwóch komponentów: Result oraz Keypad. Pierwszy komponent posłuży do wyświetlenia wyniku, a drugi do wyświetlenia klawiatury.



Rys. 6.6 Wygląd interfejsu aplikacji Kalkulator z zaznaczonymi komponentami

W katalogu *src* należy utworzyć folder o nazwie *components*, w którym będą przechowywane komponenty. W tym miejscu tworzymy dwa pliki o nazwach Results.js i Keypad.js. Komponent Result będzie zawierał kod z listingu 6.8.

Listing 6.8 *Kod komponentu Result*

```
function Result(props) {
  let { result } = props
  return (
    <div className="result">
      <input type="text" value={result}></input>
    </div>
  )
}
export default Result
```

Komponent klawiatury będzie zawierał zestaw wielu przycisków, które po ich naciśnięciu będą wykonywały odpowiednią operację. Implementacja tego komponentu będzie wyglądała tak jak na listingu 6.9.

Listing 6.9 *Kod komponentu Keypad*

```
function KeyPad(props) {
  return (
    <div className="button">
      <button className="primary" name="(" onClick={
        e => props.onClick(e.target.name)}>(</button>
      <button className="primary" name=")" onClick={
        e => props.onClick(e.target.name)}>)</button>
      <button className="primary" name="C" onClick={
        e => props.onClick(e.target.name)}>C</button>
```

```
        <button className="primary" name="CE" onClick={
          e => props.onClick(e.target.name)}>CE</button><br />
        ...
      </div>
    )
}
export default KeyPad
```

Zdarzenie kliknięcia komponentu musi być wysłane do komponentu nadrzędnego, z informacją o tym, który przycisk został kliknięty. W tym celu przy każdym naciśnięciu przycisku wywoływana jest funkcja *props.onClick()* i przekazywany jest do niej *e.target.name* jako argument.

W komponencie głównym *App*, który jest rodzicem komponentów potomnych *Result* i *Keypad*, należy umieścić te komponenty. W pliku App.js należy zdefiniować funkcje dla podstawowych możliwości kalkulatora. Będą to trzy główne funkcje:

- *calculate* – obliczająca wynik wyrażenia; wywoływana po wciśnięciu przycisku "=",
- *reset* – czyszcząca dane wyjściowe; wyzwalana po naciśnięciu "C",
- *backspace* – usuwająca ostatni znak, który został wciśnięty; wyzwalana po wciśnięciu "CE".

Listing 6.10 *Kod komponentu głównego App*

```
import { useState } from 'react'
import Result from './components/Result'
import Keypad from "./components/Keypad"
import './App.css'

function App() {
 const [state, setState] = useState({ result: "" })
 const onClick = button => {
  switch (button) {
   case "=":
    calculate()
    break
   case "C":
    reset()
    break
   case "CE":
    backspace()
    break
   default:
    setState({ result: state.result + button })
  }
 }
 const calculate = () => {
  try {
   setState({
    result: (eval(state.result) || "") + ""
   })
```

```
    } catch (e) {
      setState({
        result: "error"
      })
    }
  }
  const reset = () => {
    setState({
      result: ""
    })
  }
  const backspace = () => {
    setState({
      result: state.result.slice(0, -1)
    })
  }
  return (
    <div>
      <div className="srodek">
        <h3>Kalkulator</h3>
        <Result result={state.result} />
        <Keypad onClick={onClick} />
      </div>
    </div>
  )
}
export default App
```

Na koniec należy jeszcze ostylować komponenty, żeby wyglądały jak na rysunku 6.6. Na listingu 6.11 znajduje się kod arkusza stylów.

Listing 6.11 *Arkusz stylów App.css*

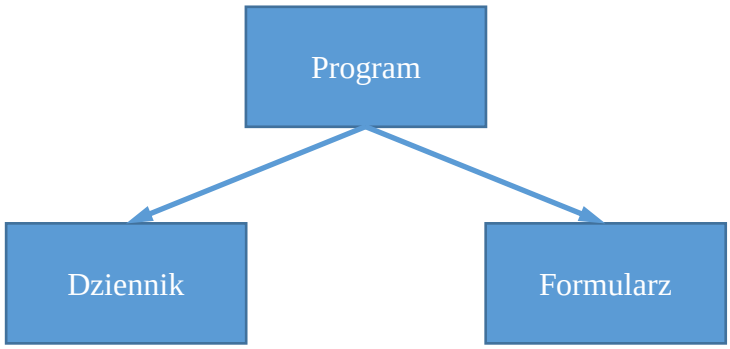
```
.button{
  padding:10px;
}
.result{
  padding: 0 10px 0 10px;
}
input {
  padding: 10px;
}
.srodek {
  text-align: center;
}
button {
  border-radius: 5px;
  font-family: 'Open Sans', sans-serif;
  font-size: 18px;
  font-weight: bold;
```

```
  letter-spacing: 1px;
  max-width: 40px;
  outline: 10;
  width: 80%;
}
button.primary {
  background: #f58733;
}
button.secondary {
  background: #c1d7aa;
}
```

Zadanie 6.6. Aplikacja *Lista zadań* wykorzystująca kontrolowany formularz oparty na komponentach funkcyjnych

Implementacja aplikacji składającej się z formularza, która wyświetla pod nim tabelę z listą dodawanych prac do wykonania.

Użytkownicy będą wypełniać pola w komponencie *Formularz*, aby zarejestrować wykonane przez siebie zadanie. Wprowadzone zadanie będzie przekazane do głównego komponentu *Program*, aby następnie mogło zostać przekazane do komponentu *KartaPrac*, gdzie z kolei zostanie wyświetlone pod formularzem. Aby formularz był kontrolowany należy użyć *State* w komponencie *Formularz*, jako pojedynczego źródła informacji dla zapisów w *KartaPrac*.



Rys. 6.7 Struktura aplikacji

***Listing 6.12** Implementacja formularza bez zastosowania stanów*

```
function Formularz() {
  const handleSubmit = (e) => {
    e.preventDefault()
  }
  return (
    <form onSubmit={e => { handleSubmit(e) }}>
      <label>Opis pracy:</label> <br />
      <input name='opis' type='text' /> <br />
      <label>Nazwa:</label> <br />
      <input name='nazwa' type='text' /> <br />
      <label>Date:</label> <br />
```

```
    <input name='data' type='date' /> <br />
    <label>Priorytet:</label> <br />
    <input type='submit' value='Dodaj zadanie' />
   </form>
  )
}
export default Formularz
```

Używając kontrolowanych formularzy, wartość wejścia jest ustawiana wartością przechowywaną w stanie. Stan jest aktualizowany za każdym razem, gdy użytkownik dokona zmiany w jednym z wejść formularza, a nie tylko wtedy, gdy użytkownik naciśnie przycisk *submit*. W rezultacie, formularze stają się dynamiczne.

Następnie z Reacta zostanie zaimportowany hook *useState*, dzięki któremu będzie można w tym funkcjonalnym komponencie przechowywać stan.

Podstawowa składnia użycia hooka *useState* jest następująca:

```
const [state, setState] = useState()
```

Po lewej stronie, *state* jest nazwą obiektu stanu, natomiast *setState* jest funkcją, której używa się do ustawienia stanu tego obiektu. Po prawej stronie, za pomocą *useState()* - można ustawić wartość początkową stanu (np. useState(0) jeśli wartość początkowa ma wynosić 0).

Dla każdego indywidualnego wejścia należy dodać zdarzenie *onChange*, aby zaktualizować stan na każdym elemencie *input* za pomocą funkcji zaczepu *useState*. Poza tym trzeba przekazać cel zdarzenia, którego wartość zawiera informacje wejściowe od użytkownika. Oto przykład elementu input dla opisu:

```
<input name='opis' type='text' value={opis} onChange={e => ustawOpis(e.target.value)} />
```

W tym momencie należy upewnić się, że każde wejście posiada właściwość *value*, która jest ustawiona na odpowiadający mu obiekt stanu. Do powyższego kodu, dodajemy następującą linię, pomiędzy znacznikami otwierającymi i zamykającymi dla elementu *input*:

```
value={opis}
```

Do konfiguracji komponentu głównego *Program* (zmieniona nazwa z App) niezbędne będą następujące elementy:

- Obiekt stanu należący do App, który przechowuje wszystkie zadania do wykonania i może być przkazany w dół hierarchii komponentów jako właściwość props.
- Funkcję, która przyjmie nowe zadanie i doda go do stanu. Ta funkcja zostanie przekazana do komponentu formularza jako props.

Listing 6.13 *Kod komponentu Program*

```
import { useState } from 'react'
import KartaPrac from './components/KartaPrac'
import Formularz from './components/Formularz'
import 'bootstrap/dist/css/bootstrap.css';
function Program() {
```

```
  const [dziennikZadan, ustawDziennikZadan] = useState([])
  const dodajPrace = (zadanie) => {
    let zadania = [...dziennikZadan, zadanie]
    ustawDziennikZadan(zadania)
  }
  return (
    <section>
      <Formularz dodajPrace={dodajPrace} />
      <KartaPrac dziennik={dziennikZadan} />
    </section>
  )
}

export default Program
```

W komponencie *Program* należy teraz wykonać dwie czynności:

- użyć destrukturyzacji w celu uzyskania dostępu do funkcji *dodajZadanie*, która została przekazana przez komponent *Program*:

  ```
  function Formularz({ dodajZadanie }) {
  ```

- przekazać obiekty stanu do funkcji *dodajZadanie* wewnątrz *handleSubmit*, która jest wywoływana, gdy użytkownik naciśnie przycisk "Dodaj zadanie" dla formularza.

  ```
  dodajZadanie([opis, nazwa, data, priorytet])
  ```

Listing 6.14 *Ostateczna wersja komponentu formularz*

```
import { useState } from 'react'
import './Formularz.css'

function Formularz({ dodajPrace }) {
   const [opis, ustawOpis] = useState('')
   const [nazwa, ustawNazwe] = useState('')
   const [data, ustawDate] = useState('')

   const handleSubmit = (e) => {
      dodajPrace([opis, nazwa, data])
      e.preventDefault()
   }

   return (
      <form onSubmit={handleSubmit} class="col-lg-6 offset-lg-3">
         <div class="form-group mb-3">
            <label class="form-label">Nazwa</label>
            <input type="text" name="nazwa" class="form-control"
               value={nazwa} onChange={e => ustawNazwe(e.target.value)} />
         </div>
         <div class="form-group mb-3">
            <label class="form-label">Opis pracy</label>
            <input type="text" name="opis" class="form-control"
```

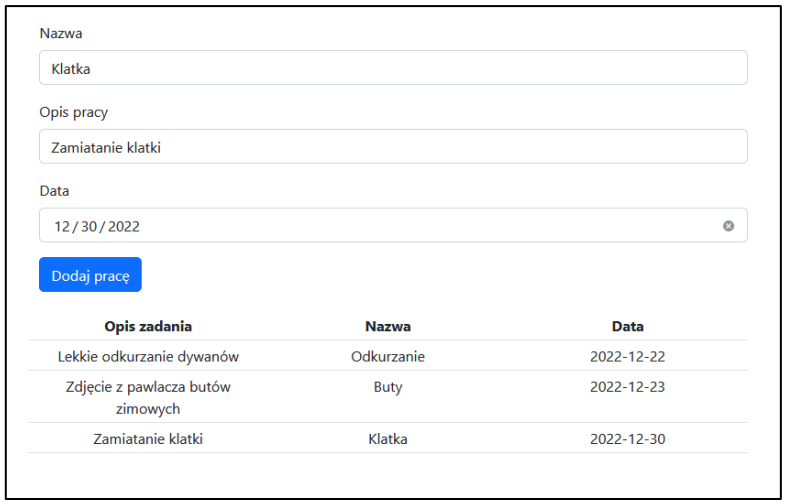
```
            value={opis} onChange={e => ustawOpis(e.target.value)} />
        </div>
        <div class="form-group mb-3">
          <label class="form-label">Data</label>
          <input type="date" name="data" class="form-control"
            value={data} onChange={e => ustawDate(e.target.value)} />
        </div>
        <input type="submit" name="submit" class="btn btn-primary" value="Dodaj pracę" />
      </form>
  )
}
export default Formularz
```

W komponencie *KartaPrac*, to należy użyć funkcji map i utworzyć wiersz tabeli dla każdego generowanego zadania do wykonania.

Listing 6.15 *Kod komponentu KartaPrac*

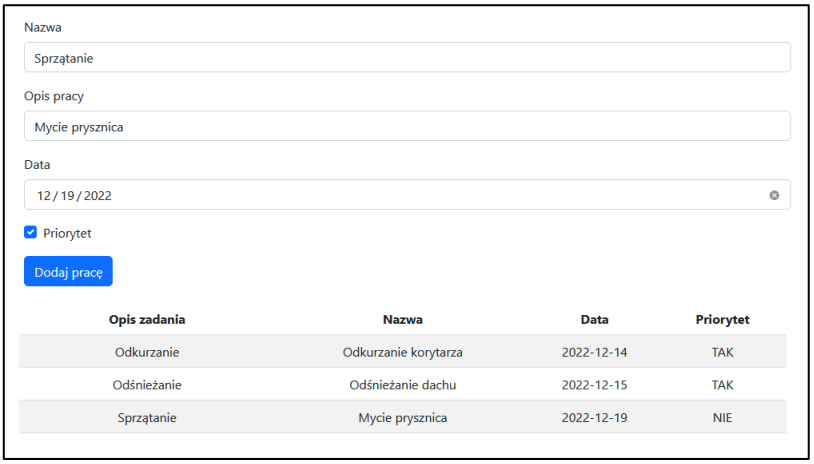
```
import './KartaPrac.css'

function KartaPrac(props) {
  return (
    <table class="table table-sm">
      <thead>
        <tr class="row">
          <th class="col-sm-2" scope="col">Opis zadania</th>
          <th class="col-sm-2" scope="col">Nazwa</th>
          <th class="col-sm-2" scope="col">Data</th>
        </tr>
      </thead>
      <tbody>
        {props.dziennik.map((v, k) => {
          return (<tr class="row" key={k}>
            <td class="col-sm-2">{v[0]}</td>
            <td class="col-sm-2">{v[1]}</td>
            <td class="col-sm-2">{v[2]}</td>
          </tr>)
        })}
      </tbody>
    </table>
  )
}
export default KartaPrac
```

W efekcie otrzymamy aplikację z formularzem oraz znajdującą się pod nim listą prac do wykonania (Rys. 6.8).



Rys. 6.8 Wygląd aplikacji po wykonaniu powyższych działań

Uzupełnij aplikację o element checkbox, który dodaje informację, czy praca jest priorytetowa. Wynik uzyskany po modyfikacji powinien być podobny do zrzutu ekranowego z Rys. 6.9.



Rys. 6.9 Ostateczny wygląd aplikacji z komponentem checkbox

Zadanie 6.7. Zarządzanie stanem komponentów w React

- Utworzenie wstępnej wersji aplikacji przedstawionej na Rys. 6.10, zawierającej jeden główny komponent App (Listing 6.16) ostylowany wg arkusza z listingu 6.17

Listing 6.16 *Kod komponentu głównego App*

```
import './App.css';
import React from "react"

function App() {
 return (
  <div className="grid-parent">
   <div className="header">
    <h1>Nagłówek</h1>
```

```
      <p>Aktualny rozmiar czcionki: <strong>x</strong></p>
      <p>Aktualny kolor czcionki: <strong>x</strong></p>
      <p>
        Liczba lajków: <strong>0</strong>
      </p>
    </div>
    <div className="sidebar">
      <input type="text" />
      <input type="text" />
      <button>Ustaw parametry tekstu na 20px i pink.</button>
    </div>
    <div className="main-area">
      <p>Szkielety programistyczne w aplikacjach internetowych: Node, MongoDB, Express, React.</p>
    </div>
    <footer className="footer">
      <p>
        Stopka <button>Ustaw parametry tekstu na 30px, a kolor pozostaw bez zmian.</button>
      </p>
      <p>
        <button>Polub tę stronę!</button>
      </p>
    </footer>
  </div>
 )
}

export default App
```

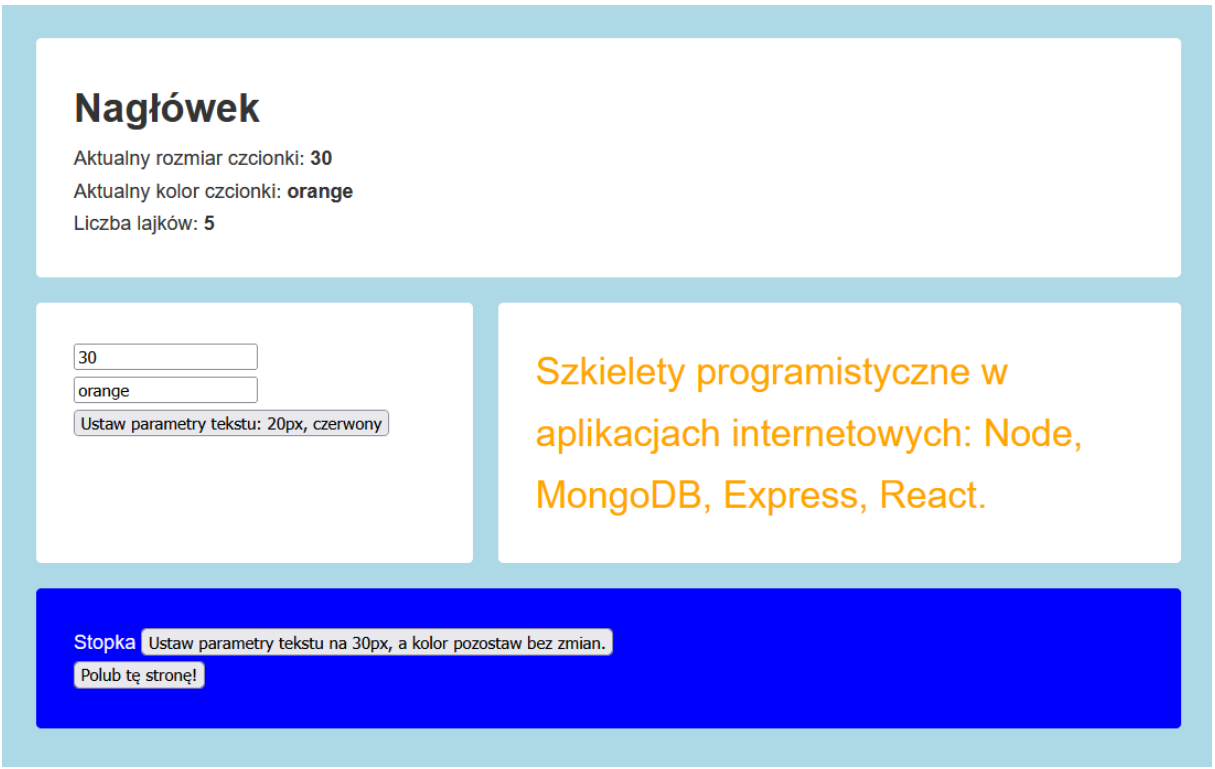
Listing 6.17 *Arkusz stylów komponenetu głównego*

```
body {
 font-family: sans-serif;
 color: #333;
 padding: 30px;
 margin: 0;
 background-color: lightblue;
 line-height: 1.65;
}

h1,
p {
 margin: 0;
}

.grid-parent {
 display: grid;
 gap: 20px;
}

@media screen and (min-width: 768px) {
```

```
  .grid-parent {
    display: grid;
    grid-template-columns: 350px 1fr;
  }

  .header,
  .footer {
    grid-column: 1 / 3;
  }
}

.header,
.sidebar,
.main-area {
  background-color: #fff;
  padding: 30px;
  border-radius: 4px;
}

.footer {
  background-color: blue;
  color: #fff;
  padding: 30px;
  border-radius: 4px;
}
```

Rys. 6.10 Wygląd początkowy aplikacji

- Ustalenie początkowego stanu: rozmiaru (na 18px) i koloru czcionki (na zielony)
- Wyświetlenie wartości ustawionych parametrów w nagłówku
- Ostylowanie tekstu znajdującego się w oknie głównym strony wyżej ustawionymi parametrami
- Wyświetlenie konfiguracji czcionki w polach tekstowych lewego paska
- Implementacja funkcji obsługujących zmianę wartości w polach tekstowych lewego paska i powodujących zmianę stylu czcionki okna głównego
- Zaprogramowanie funkcji obsługującej przycisk znajdujący się w lewym pasku, który zmienia styl wyświetlanego tekstu w oknie głównym strony
- Zaprogramowanie zdarzenia kliknięcia na przycisku *Ustaw parametry tekstu na 30px, a kolor pozostaw bez zmian.* znajdującego się w stopce strony
- Oprogramowanie przycisku Pollub tę stronę! umiejscowionego w stopce oraz wyświetlenie liczby lajków w nagłówku



- Dekompozycja komponentu głównego na komponenty podrzędne: Header, Sidebar, MainArea, Footer w strukturze projektu src/components oraz wykorzystanie ich w komponencie głównym App