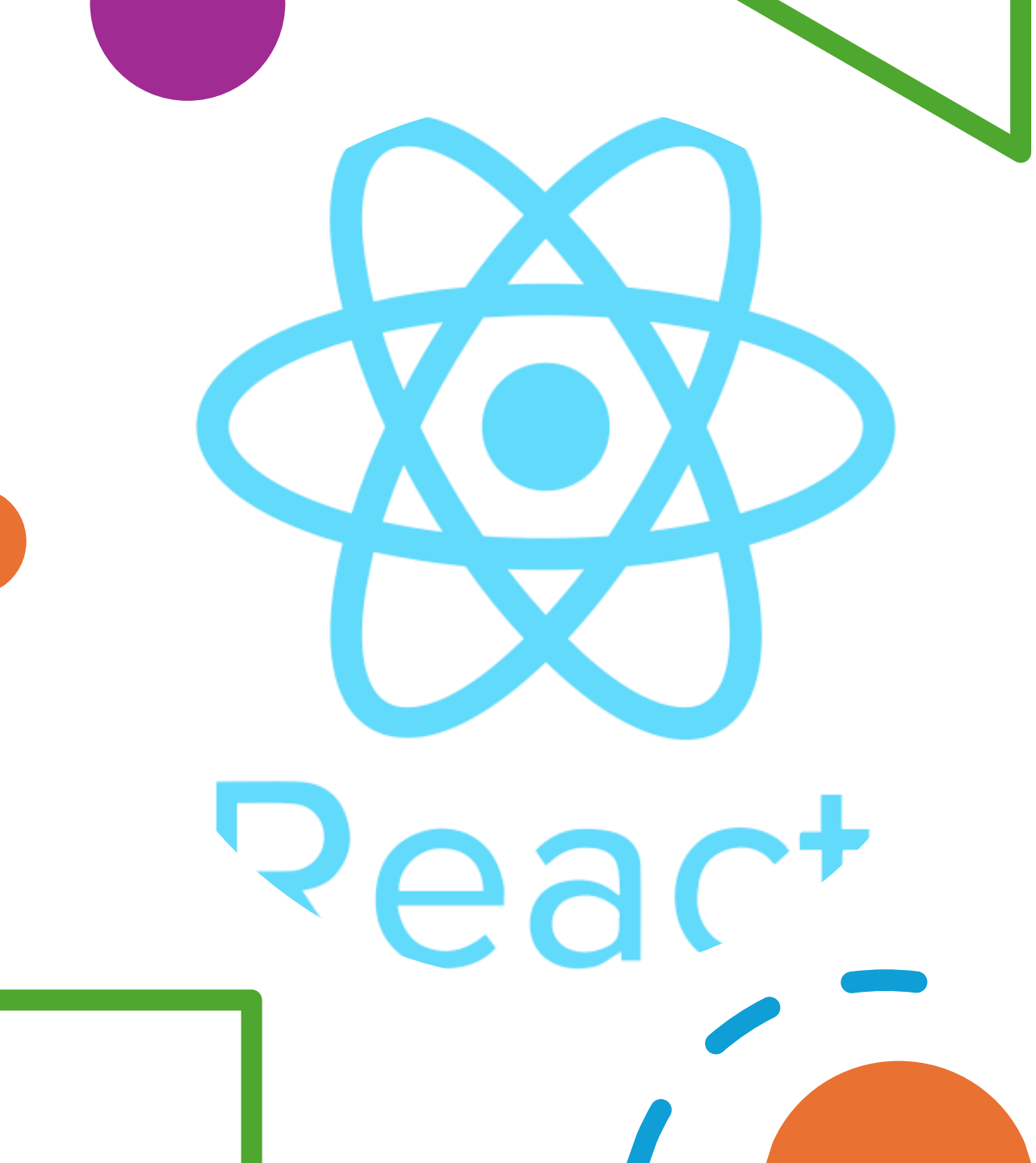


React od podstaw

Zbigniew Kluczkowski

Spis treści

Czym jest React

React to **biblioteka JavaScript** stworzona przez Facebooka, służąca do budowania **interfejsów użytkownika (UI)**. Kluczowe cechy to:

- **Komponenty:** Aplikacje w React są zbudowane z małych, niezależnych i wielokrotnego użytku bloków kodu, zwanych komponentami.
- **Jednokierunkowy przepływ danych:** Dane w React płyną w dół, od komponentu rodzica do komponentu dziecka.
- **Wirtualny DOM:** Zamiast bezpośrednio manipulować rzeczywistym DOM (który jest powolny), React używa jego wirtualnej kopii. Dzięki temu renderowanie stron jest niezwykle szybkie i wydajne.

Instalacja i tworzenie nowego projektu

Aby zacząć pracę z React, potrzebujesz zainstalować **Node.js** i **npm** (menedżer pakietów Node.js). Po ich instalacji, najprostszym sposobem na stworzenie nowego projektu jest użycie narzędzia takiego jak **Vite** (zalecany) lub **Create React App**.

Krok 1: Sprawdź Node.js i npm

Otwórz terminal i wpisz:

```
node -v
npm -v
```

Krok 2: Utwórz nową aplikację

Użyj Vite, aby szybko stworzyć nowy projekt.

```
npm create vite@latest my-react-app -- --template react
```

Krok 3: Przejdź do folderu projektu

```
cd my-react-app
```

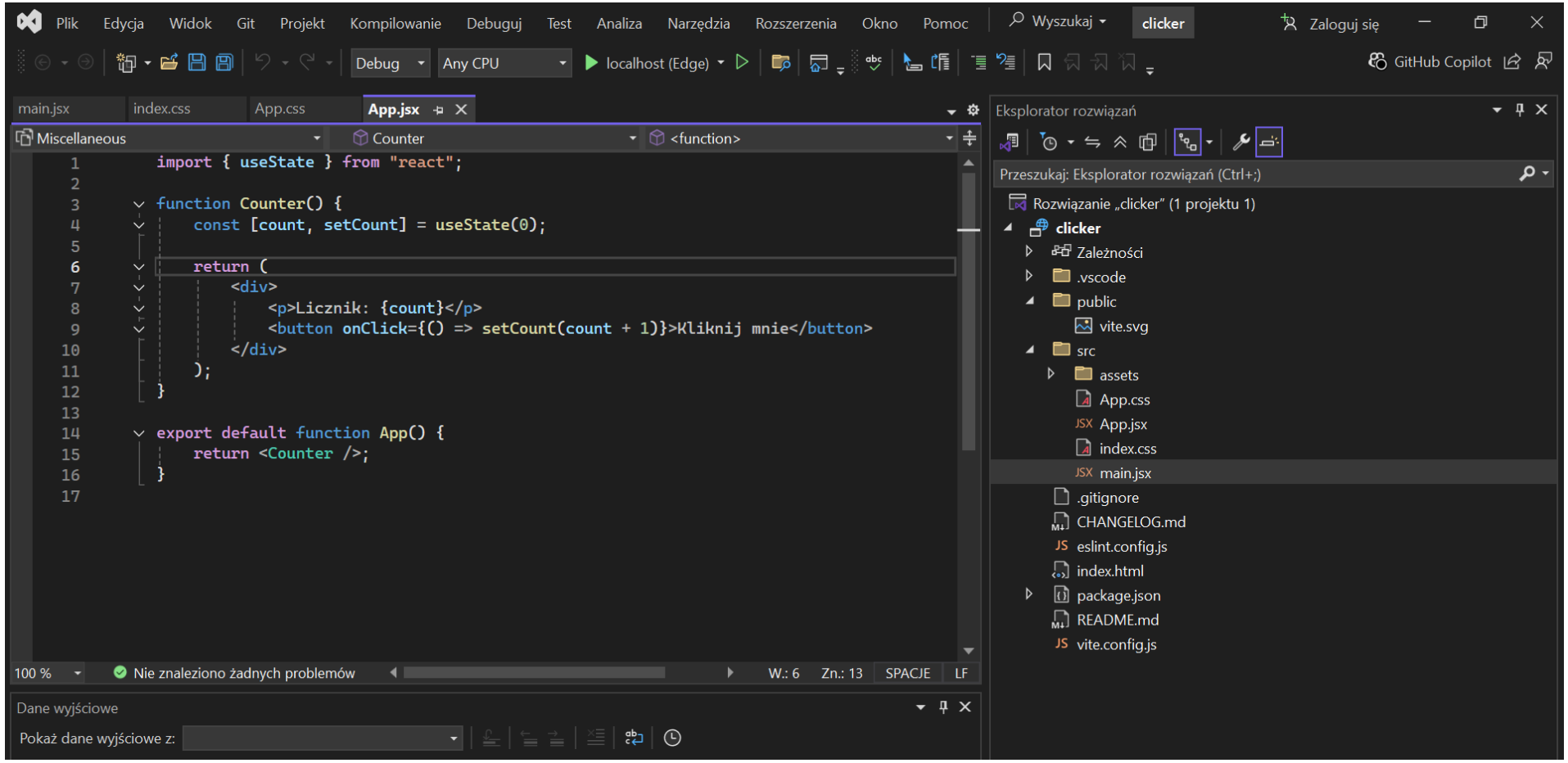
Instalacja i tworzenie nowego projektu

Krok 4: Zainstaluj zależności i uruchom

```
npm install
npm run dev
```

Problemy?
React potrzebuje wolnego portu. Problemy sprawia również zapora systemowa lub przeglądarka. Visual Studio czasem „grymasi” podczas uruchamiania projektu. Co wtedy? Najlepiej uruchamiać projekt ręcznie korzystając z eksploratora plików i przeglądarki. Efekt ten sam. Stresu mniej.

Struktura aplikacji w VisualStudio

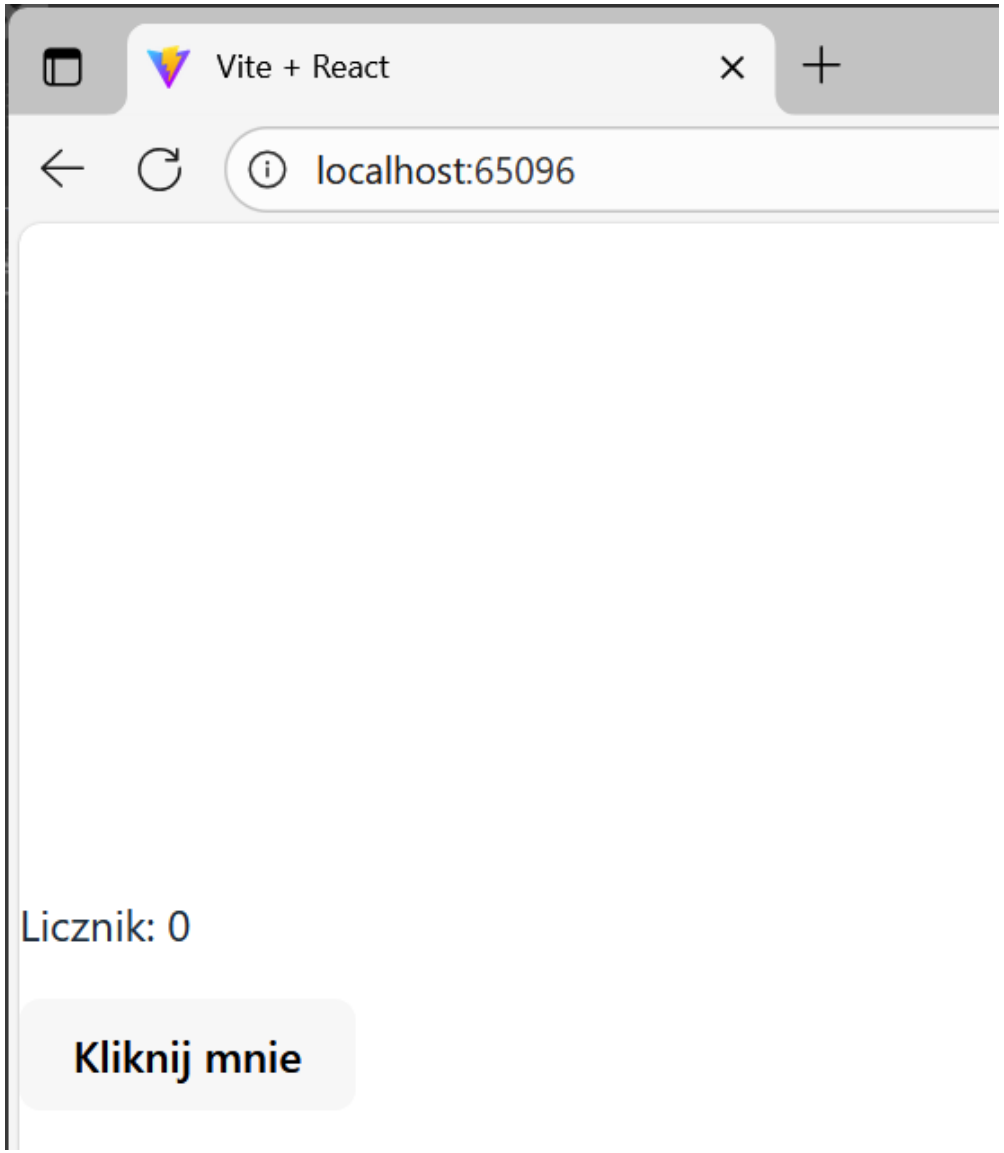


Typowa struktura plików

```
my-app/
├── node_modules/        # pakiety npm, biblioteki (nie edytujemy)
├── public/              # pliki publiczne, dostępne bezpośrednio w przeglądarce
│   ├── index.html       # główny plik HTML
│   └── favicon.ico      # ikona aplikacji
├── src/                 # kod źródłowy React
│   ├── assets/          # obrazy, fonty, pliki statyczne
│   ├── components/      # komponenty React
│   │   └── MyButton.jsx
│   ├── App.jsx          # główny komponent aplikacji
│   ├── main.jsx         # plik wejściowy (Vite) lub index.js (CRA)
│   └── index.css        # globalne style CSS
├── package.json         # zależności, skrypty npm
├── package-lock.json    # blokada wersji pakietów
└── vite.config.js / react-scripts/ # konfiguracja Vite lub CRA
```

Widok w przeglądarce

Domyślnie React przydziela sobie port 3000, ale uruchamiając projekt w Visual Studio można zobaczyć taki widok.



Moduły

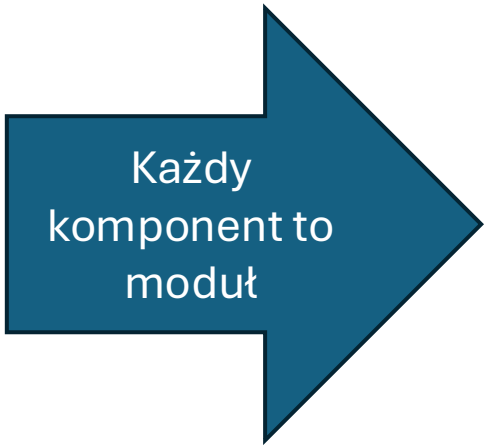
Moduły to **oddzielne pliki z kodem**, które możemy **importować i wykorzystywać w innych częściach projektu**.
Dzięki modułom aplikacja nie musi być jednym wielkim plikiem — możemy ją dzielić na mniejsze, czytelniejsze części.

Bez modułów wszystko byłoby w jednym pliku — trudno byłoby się w tym odnaleźć. Dzięki modułom:

- kod jest **bardziej uporządkowany**,
- łatwiej go **ponownie wykorzystać**,
- można **dzielić aplikację na logiczne części** (np. komponenty, funkcje, style, dane),
- łatwiej się współpracuje w zespole.

Moduły

```
export default function Hello() {
  return <h1>Cześć, jestem komponentem!</h1>;
}
```




```
import Hello from "./Hello";

function App() {
  return (
    <div>
      <Hello />
    </div>
  );
}

export default App;
```

Komponenty

Komponenty to serce każdej aplikacji React. Mogą to być **funkcje** (preferowane w nowoczesnym React) lub **klasy**.

```
// src/components/WelcomeMessage.jsx

function WelcomeMessage() {
  return (
    <div>
      <h1>Witaj w mojej prezentacji!</h1>
      <p>To jest prosty komponent React.</p>
    </div>
  );
}

export default WelcomeMessage;
```


```
import WelcomeMessage from
'./components/WelcomeMessage';

function App() {
  return (
    <div>
      <WelcomeMessage />
    </div>
  );
}

export default App;
```

Propsy (właściwości)

Props (skrót od "properties") to sposób na **przekazywanie danych z komponentu rodzica do komponentu dziecka**. Umożliwiają tworzenie dynamicznych i wielokrotnego użytku komponentów.

Stwórz komponent, który przyjmuje name jako prop:

```
// src/components/Greeting.jsx

function Greeting(props) {
  return (
    <h1>Witaj, {props.name}!</h1>
  );
}

export default Greeting;
```


```
import Greeting from './components/Greeting';

function App() {
  return (
    <div>
      <Greeting name="Anna" />
      <Greeting name="Paweł" />
    </div>
  );
}
```

Typy propsów

Typ propsa	Przykład w użyciu	Przykład wartości
Tekst	<Hello name="Ania" />	"Ania"
Liczba	<Score points={10} />	10
Tablica	<List items={['a','b']} />	['a','b']
Obiekt	<UserCard user={user} />	{name:'Jan', age:25}
Funkcja	<Button onClick={handleClick} />	() => alert('Klik')
Children	<Box> Tekst </Box>	Wnętrze komponentu

Hook

W React — hook (czyt. huk) to specjalna funkcja, która pozwala używać funkcjonalności Reacta (np. stanu, efektów) w komponentach funkcyjnych (czyli takich tworzonych jako zwykłe funkcje — nie klasy).

Dlaczego powstały hooki?

Dawniej, aby mieć np. stan lub cykl życia komponentu, trzeba było używać klas (class components). Hooki pozwoliły robić to samo w komponentach funkcyjnych, które są prostsze i czytelniejsze.

Hook

Kategoria	Nazwa Hooka	Opis działania
Podstawowe	useState	Przechowuje i aktualizuje stan w komponencie funkcyjnym.
	useEffect	Wykonuje efekty uboczne (np. pobieranie danych, zmiana tytułu strony).
	useContext	Umożliwia dostęp do wartości z React Context bez przekazywania propsów.
Zaawansowane	useReducer	Zarządza złożonym stanem (alternatywa dla useState, np. mini Redux).
	useCallback	Zapamiętuje funkcję, aby nie była tworzona ponownie przy każdym renderze.
	useMemo	Zapamiętuje wynik kosztownego obliczenia.
	useRef	Przechowuje odniesienie do elementu DOM lub wartości, która nie powoduje ponownego renderowania.
	useImperativeHandle	Pozwala kontrolować, jakie metody i dane są dostępne przez ref.
	useLayoutEffect	Działa jak useEffect, ale synchronizuje się **przed** malowaniem komponentu (używane np. w animacjach).
	useDebugValue	Dodaje etykietę w React DevTools dla niestandardowych hooków.
Optymalizacja i synchronizacja (React 18+)	useTransition	Oznacza część aktualizacji jako „niepilną”, np. przy ładowaniu danych.
	useDeferredValue	Opóźnia aktualizację wartości, poprawiając płynność UI.
	useId	Generuje unikalne identyfikatory (np. dla pól formularza).
	useSyncExternalStore	Synchronizuje komponenty z zewnętrznym źródłem danych (np. Redux).
	useInsertionEffect	Działa **przed renderem**, używany głównie przez biblioteki CSS-in-JS.
Eksperymentalne (React 19+)	use	Pozwala bezpośrednio korzystać z obiektów Promise lub kontekstów.
	useOptimistic	Umożliwia optymistyczne aktualizacje UI (np. pokazanie wyniku przed odpowiedzią serwera).

useState (Hook)

useState to jeden z najważniejszych **Hooków** w React.

Czym jest stan?

Stan to dane, które komponent "pamięta" i które, po zmianie, powodują ponowne wyrenderowanie komponentu.

```
Import React, { useState } from 'react';


function Counter() {
  // Deklaracja zmiennej stanu o nazwie 'count' z wartością początkową 0
  const [count, setCount] = useState(0);


  const increment = () => {
    // Aktualizacja stanu. Zmiana wywoła re-render komponentu.
    setCount(count + 1);
  };


  return (
    <div>
      <p>Liczba kliknięć: {count}</p>
      <button onClick={increment}>Kliknij mnie</button>
    </div>
  );
}
```

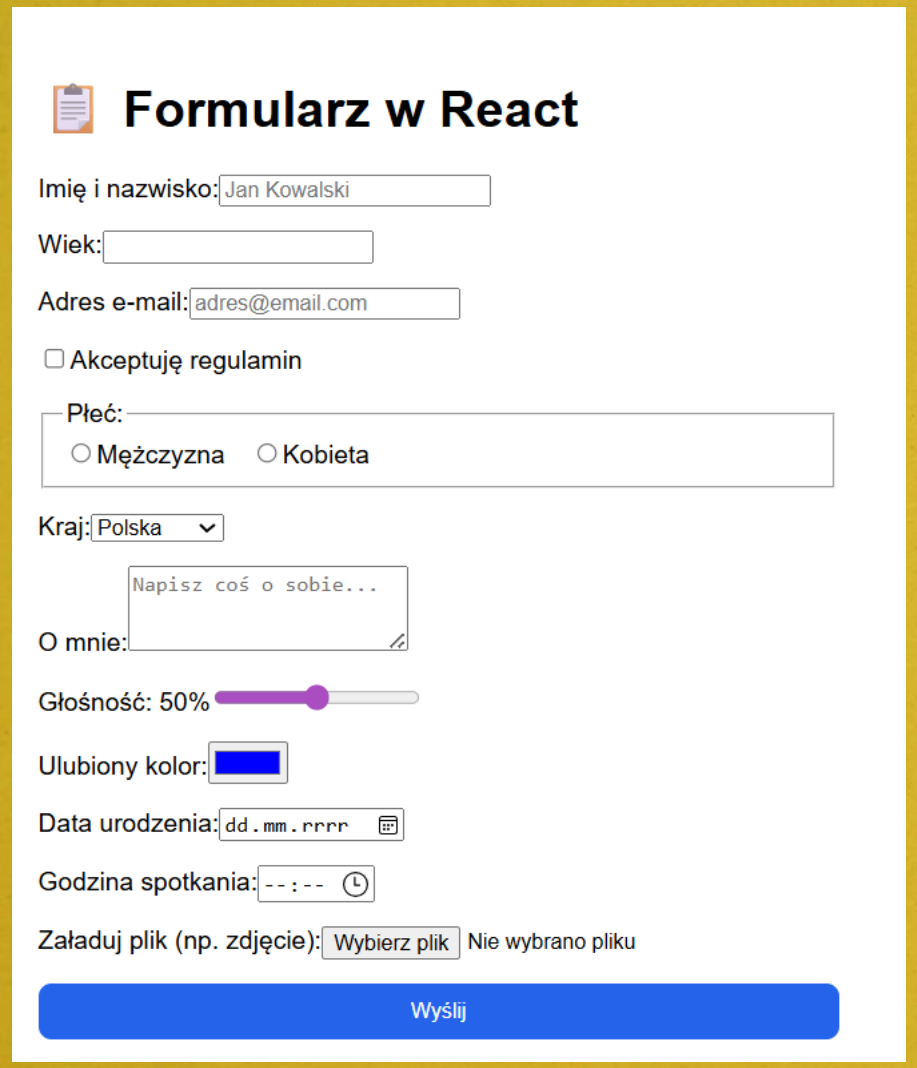
useEffect

Co to są „efekty uboczne”?

Efekty uboczne to wszystko, co dzieje się **poza renderowaniem komponentu**, np.:

- pobieranie danych z API,
- zapisywanie do localStorage,
- subskrypcje, timery (setInterval, setTimeout),
- reagowanie na zmianę stanu lub propsów,
- aktualizacja tytułu strony (document.title).

```
import { useEffect, useState } from "react";

export default function Users() {
  const [users, setUsers] = useState([]);

  useEffect(() => {

fetch("https://jsonplaceholder.typicode.com/users")
    .then(res => res.json())
    .then(data => setUsers(data));
  }, []);

  return (
   <ul>
    {users.map(u => <li key={u.id}>{u.name}</li>)}
   </ul>
  );
}
```

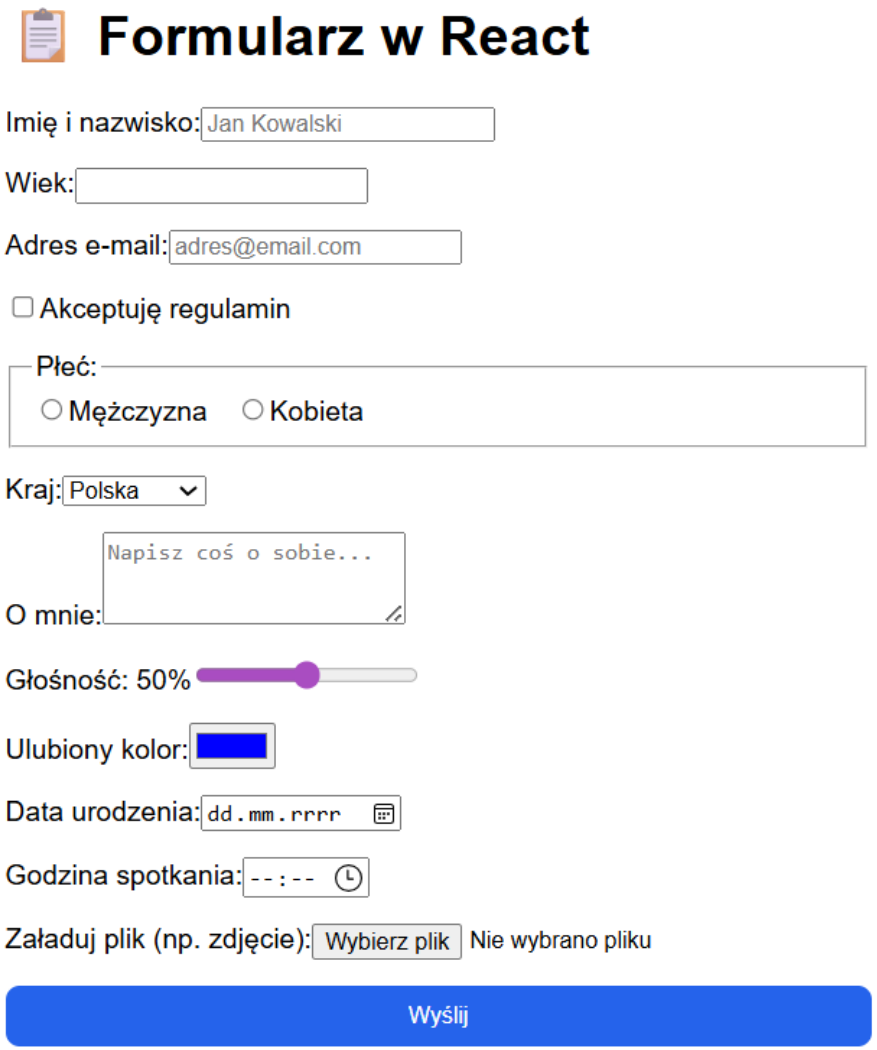


Przykładowy formularz z popularnymi widżetami

```jsx
import React, { useState } from "react";
export default function App() {

  const [formData, setFormData] = useState({
    name: "",
    age: "",
    email: "",
    agree: false,
    gender: "",
    country: "Polska",
    bio: "",
    volume: 50,
    favoriteColor: "#0000ff",
    birthDate: "",
    meetingTime: "",
    file: null,
    filePreview: null,
  });
  const handleChange = (e) => {
    const { name, type, value, checked, files } =
e.target;
    if (type === "checkbox") {
      setFormData({ ...formData, [name]: checked
});
    } else if (type === "file") {
      const file = files[0];
      setFormData({
        ...formData,
        file,
        filePreview: file ? URL.createObjectURL(file)
: null,
      });
    } else {
      setFormData({ ...formData, [name]: value });
    }
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    alert("Dane przesłane:\n" +
JSON.stringify(formData, null, 2));
  };
  return (
    <div style={{ padding: "20px", fontFamily: "Arial,
sans-serif" }}>
      <h1>📄 Formularz w React</h1>
      <form
        onSubmit={handleSubmit}
        style={{
          display: "flex",
          flexDirection: "column",
          gap: "15px",
          maxWidth: "500px",
        }}
      >
        {/* Imię i nazwisko */}
        <label>
          Imię i nazwisko:
          <input
            type="text"
            name="name"
            value={formData.name}
            onChange={handleChange}
            placeholder="Jan Kowalski"
          />
        </label>
        {/* Wiek */}
        <label>
          Wiek:
          <input
            type="number"
            name="age"
            value={formData.age}
            onChange={handleChange}
            min="0"
          />
        </label>
        {/* Email */}
        <label>
          Adres e-mail:
          <input
            type="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
            placeholder="adres@email.com"
          />
        </label>
        {/* Checkbox */}
        <label>
          <input
            type="checkbox"
            name="agree"
            checked={formData.agree}
            onChange={handleChange}
          />
          Akceptuję regulamin
        </label>
        {/* Radio */}
        <fieldset>
          <legend>Płeć:</legend>
          <label>
            <input
              type="radio"
              name="gender"
              value="Mężczyzna"
              checked={formData.gender ===
"Mężczyzna"}
              onChange={handleChange}
            />
            Mężczyzna
          </label>
          <label style={{ marginLeft: "15px" }}>
            <input
              type="radio"
              name="gender"
              value="Kobieta"
              checked={formData.gender ===
"Kobieta"}
              onChange={handleChange}
            />
            Kobieta
          </label>
        </fieldset>
        {/* Select */}
        <label>
          Kraj:
          <select
            name="country"
            value={formData.country}
            onChange={handleChange}
          >
            <option
value="Polska">Polska</option>
            <option
value="Niemcy">Niemcy</option>
            <option
value="Francja">Francja</option>
            <option
value="Hiszpania">Hiszpania</option>
          </select>
        </label>
        {/* Textarea */}
        <label>
          O mnie:
          <textarea
            name="bio"
            value={formData.bio}
            onChange={handleChange}
            rows="3"
            placeholder="Napisz coś o sobie..."
          />
        </label>
        {/* Range */}
        <label>
          Głośność: {formData.volume}%
          <input
            type="range"
            name="volume"
            min="0"
            max="100"
            value={formData.volume}
            onChange={handleChange}
          />
        </label>
        {/* Color */}
        <label>
          Ulubiony kolor:
          <input
            type="color"
            name="favoriteColor"
            value={formData.favoriteColor}
            onChange={handleChange}
          />
        </label>
        {/* Date */}
        <label>
          Data urodzenia:
          <input
            type="date"
            name="birthDate"
            value={formData.birthDate}
            onChange={handleChange}
          />
        </label>
        {/* Time */}
        <label>
          Godzina spotkania:
          <input
            type="time"
            name="meetingTime"
            value={formData.meetingTime}
            onChange={handleChange}
          />

        </label>
        {/* File upload */}
        <label>
          Załaduj plik (np. zdjęcie):
          <input type="file" name="file"
onChange={handleChange} accept="image/*" />
        </label>
        {formData.filePreview && (
          <div style={{ marginTop: "10px" }}>
            <p>Podgląd:</p>
            <img
              src={formData.filePreview}
              alt="Podgląd pliku"
              style={{
                width: "150px",
                height: "150px",
                objectFit: "cover",
                border: "1px solid #ccc",
              }}
            />
          </div>
        )}
        <button
          type="submit"
          style={{
            padding: "10px",
            background: "#2563eb",
            color: "white",
            border: "none",
            borderRadius: "8px",
            cursor: "pointer",
          }}
        >
          Wyślij
        </button>
      </form>
      {/* Podgląd danych */}
      <div style={{ marginTop: "20px", padding:
"10px", border: "1px solid #ddd" }}>
        <h2>Podgląd danych:</h2>
        <pre>{JSON.stringify(formData, null,
2)}</pre>
      </div>
    </div>
  );
}
```

Efekt?

Najważniejsze widgety w jednym projekcie oraz przekazywanie z nich wartości.



Handlery

Handler – funkcja obsługująca zdarzenia (np. kliknięcie, wpisanie tekstu, przesłanie formularza).

Dzięki handlerom komponenty mogą reagować na akcje użytkownika.

```
function handleClick() {
  alert("Kliknięto przycisk!");
}
```

```
<button onClick={handleClick}>Kliknij
mnie</button>
```

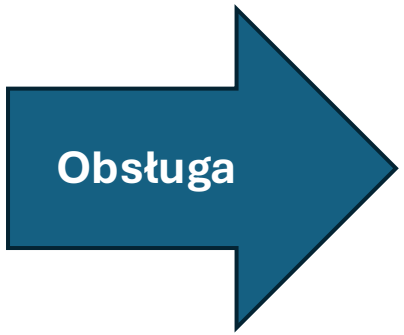
Zdarzenia mają podobne nazwy jak w DOM, ale pisane w **camelCase**.

- onClick zamiast onclick
- onChange zamiast onchange

Obsługiwane są przez **Synthetic Events** – ujednoliconą warstwę zdarzeń React.

Popularne Handlery:

- **onClick** – kliknięcie przycisku
- **onChange** – zmiana wartości inputa
- **onSubmit** – wysłanie formularza
- **onMouseEnter / onMouseLeave** – wejście/wyjście kursora
- **onKeyDown / onKeyUp** – wciśnięcie klawisza




```
import React from "react";

function App() {
  const handleClick = () => {
    console.log("Przycisk został kliknięty!");
  };

  return (
    <button
onClick={handleClick}>Kliknij</button>
  );
}

export default App;
```

Handler z parametrami

```
<button onClick={() => handleDelete(id)}>Usuń</button>
```


```
function handleDelete(itemId) {
  console.log("Usuwanie elementu:", itemId);
}
```

Handler – obsługa formularza

```
function Form() {
  const handleSubmit = (event) => {
    event.preventDefault(); // zapobiega przeładowaniu strony
    alert("Formularz wysłany!");
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" placeholder="Wpisz coś..." />
      <button type="submit">Wyślij</button>
    </form>
  );
}
```

Najlepsze praktyki:

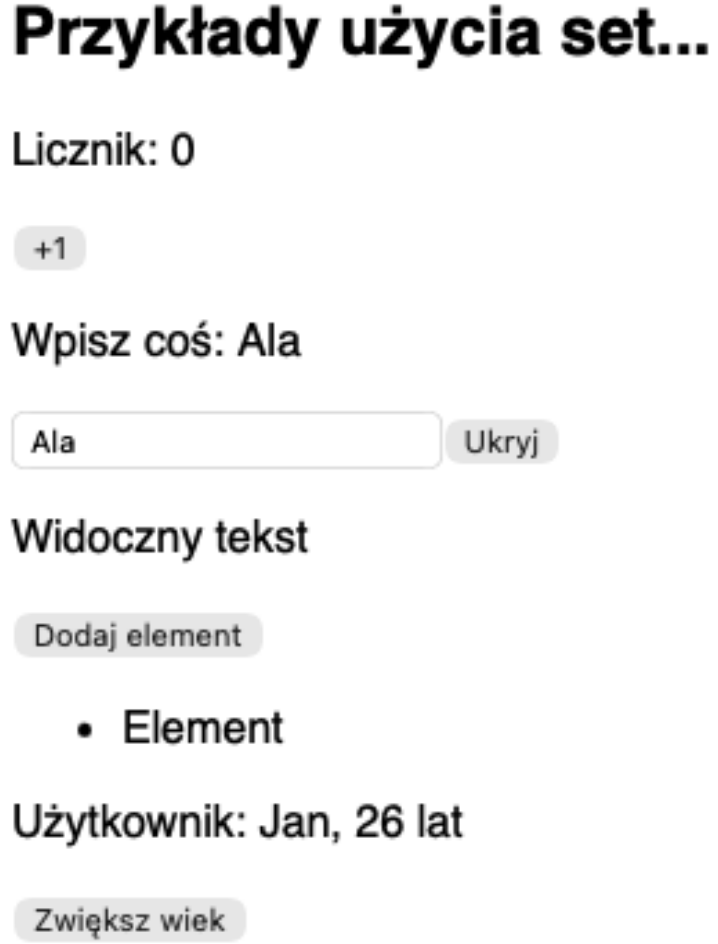
- nadaj funkcjom opisowe nazwy (handleClick, handleChange);
- korzystaj z **funkcji strzałkowych** dla prostoty;
- oddziel logikę od JSX – kod będzie czytelniejszy;
- unikaj pisania handlerów inline, jeśli są złożone.

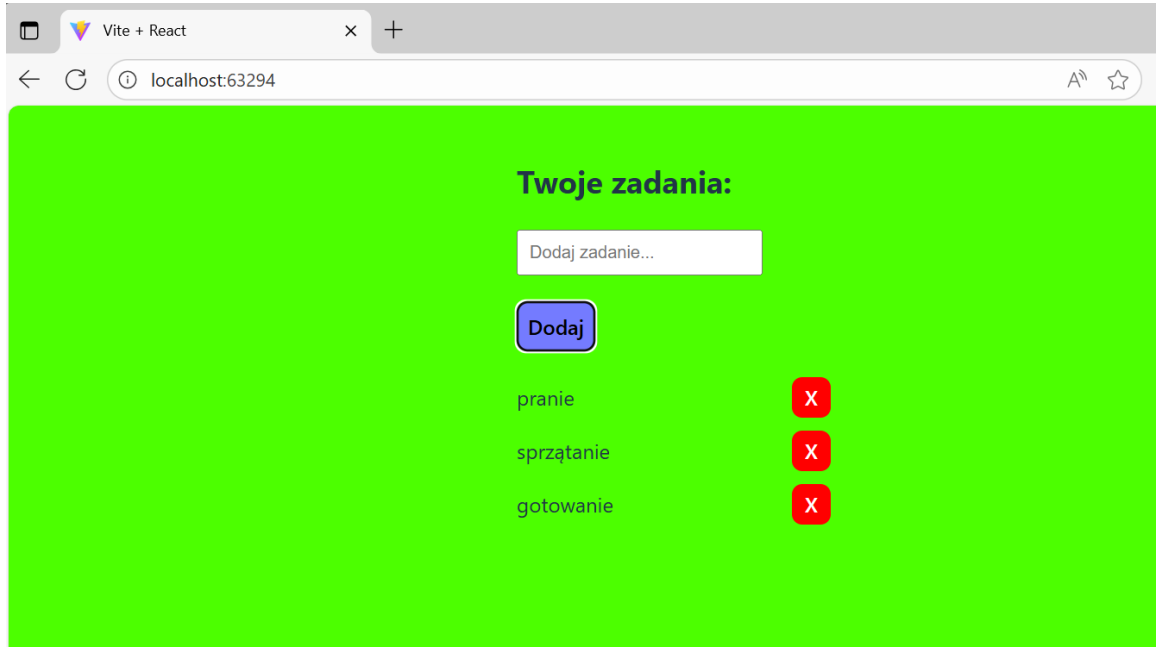
SET w React

Typ stanu	Deklaracja useState	Przykład użycia set...	Opis
Liczba	const [count, setCount] = useState(0);	setCount(count + 1); lub setCount(prev => prev + 1);	Zwiększanie licznika, np. kliknięcia.
Tekst (string)	const [text, setText] = useState("");	setText("Hello"); setText(e.target.value);	Przechowywanie danych z inputa.
Boolean (true/false)	const [isOpen, setIsOpen] = useState(false);	setIsOpen(!isOpen);	Przełącznik widoczności, checkbox, modal.
Tablica	const [items, setItems] = useState([]);	setItems([...items, "Nowy"]);setItems(items.filter(i => i !== "A"));	Dodawanie, usuwanie elementów listy.
Obiekt	const [user, setUser] = useState({ name: "Jan", age: 25 });	setUser({ ...user, age: 26 });	Aktualizacja wybranego pola w obiekcie.
Obiekt zagnieżdżony	const [form, setForm] = useState({ user: { name: "", email: "" } });	setForm({ ...form, user: { ...form.user, name: "Adam" } });	Aktualizacja pola w obiekcie wielopoziomowym.
Lista obiektów	const [tasks, setTasks] = useState([{ id: 1, text: "Zadanie" }]);	setTasks(tasks.map(t => t.id === 1 ? {...t, done: true} : t));	Modyfikowanie elementu w tablicy obiektów.

SET w React – przykłady użycia

```
import { useState } from "react";


export default function StateExamples(){

 const [count, setCount] = useState(0);

 const [text, setText] = useState("");

 const [isVisible, setIsVisible] = useState(true);

 const [items, setItems] = useState([]);

 const [user, setUser] = useState({ name: "Jan", age: 25 });



 return (

  <div style={{ padding: "20px" }}>

    <h2>Przykłady użycia set...</h2>



    {/* Licznik */}

    <p>Licznik: {count}</p>

    <button onClick={() => setCount(count + 1)}>+1</button>



    {/* Input */}

    <p>Wpisz coś: {text}</p>

    <input value={text} onChange={(e) => setText(e.target.value)} />



    {/* Boolean */}

    <button onClick={() => setIsVisible(!isVisible)}>

      {isVisible ? "Ukryj" : "Pokaż"}

    </button>

    {isVisible && <p>Widoczny tekst</p>}



    {/* Tablica */}

    <button onClick={() => setItems([...items, "Element"])}>Dodaj element</button>

    <ul>{items.map((item, i) => <li key={i}>{item}</li>)}</ul>



    {/* Obiekt */}

    <p>Użytkownik: {user.name}, {user.age} lat</p>

    <button onClick={() => setUser({ ...user, age: user.age + 1 })}>

      Zwiększ wiek

    </button>

  </div>

 );

}
```





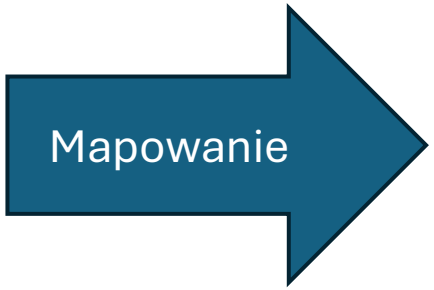
Tasks – tworzenie listy zadań

Projekt wykorzystuje Tasks oraz Mapowanie.

Umożliwia dodanie zadania, wyświetlenie oraz usunięcie za pomocą przycisku obok dodanego tekstu.

Przykładowe zastosowanie Tasks

```
// Dodawanie zadania
  function addTask() {
    if (newTask.trim() === "") return;
    setTasks([...tasks, { text: newTask, completed: false }]);
    setNewTask("");
  }
```




```
// Oznaczanie ukończonego zadania
  function toggleTask(index) {
    const updatedTasks = tasks.map((task, i) =>
      i === index ? { ...task, completed: !task.completed } : task
    );
    setTasks(updatedTasks);
  }
```

Mapowanie

W React nie zmieniamy stanu „w miejscu” – zawsze tworzymy **nową wersję tablicy** i podmieniamy starą (to się nazywa **immutability**, czyli niezmienność).

Dzięki temu React wykrywa zmiany i wie, że musi przerysować komponent.

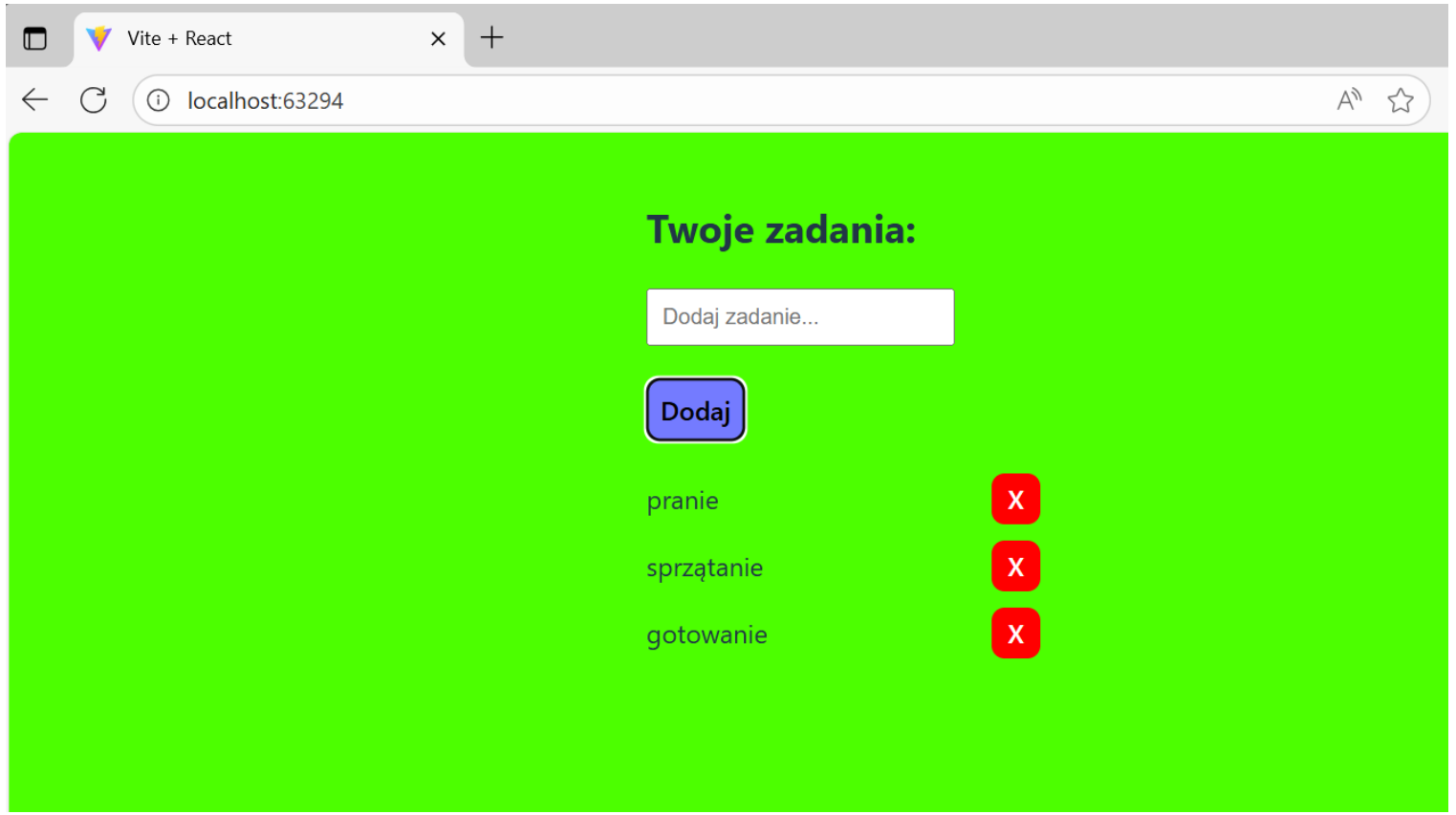
```
function toggleTask(index) {
  const updated = tasks.map((task, i) =>
    i === index ? { ...task, completed: !task.completed } : task
  );
  setTasks(updated);
}
```

Wyjaśnienie:
.map() przechodzi po każdym zadaniu.
Jeśli i === index → zmienia completed.
Jeśli nie, zwraca zadanie bez zmian.
Wynik = **nowa tablica** z podmienionym jednym elementem.
Dzięki temu nie psujesz oryginalnej tablicy, tylko masz jej nową wersję.

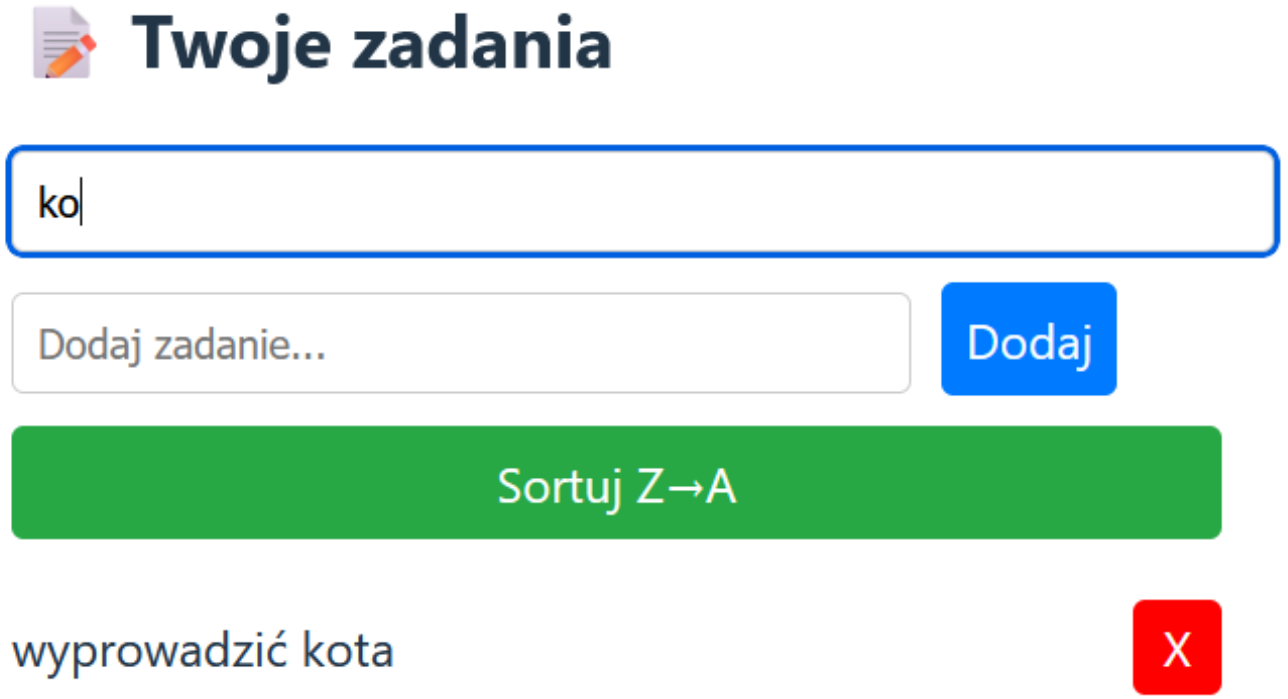
Przykład prostej Listy zadań

```
import { useState } from "react";

export default function App() {
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState("");

  // Dodawanie zadania

  function addTask() {
    if (newTask.trim() === "") return;
    setTasks([...tasks, { text: newTask, completed: false }]);
    setNewTask("");
  }

  // Usuwanie zadania

  function deleteTask(index) {
    setTasks(tasks.filter((_, i) => i !== index));
  }

  // Oznaczanie ukończonego zadania

  function toggleTask(index) {
    const updatedTasks = tasks.map((task, i) =>
      i === index ? { ...task, completed: !task.completed } : task
    );
    setTasks(updatedTasks);
  }

  return (
<div style={{ padding: "20px", maxWidth: "400px", margin: "auto" }}>
  <h2>Twoje zadania:</h2>
  <input
    type="text"
    value={newTask}
    onChange={(e) => setNewTask(e.target.value)}
    placeholder="Dodaj zadanie..."
    style={{ width: "70%", padding: "8px" }}

  />

  <button onClick={addTask} style={{ padding: "8px", marginTop: "20px" }}>
    Dodaj
  </button>
  <ul style={{ listStyle: "none", padding: 0, marginTop: "20px" }}>
    {tasks.map((task, index) => (
      <li
        key={index}
        style={{
          marginBottom: "10px",
          display: "flex",
          justifyContent: "space-between",
          alignItems: "center",
        }}

      >
        <span
          onClick={() => toggleTask(index)}
          style={{
            textDecoration: task.completed ? "line-through" : "none",
            cursor: "pointer",

          }}
        >
          {task.text}
        </span>
        <button
          onClick={() => deleteTask(index)}
          style={{ background: "red", color: "white", border: "none", padding: "5px
10px" }}
        >
          X

        </button>
      </li>
    ))}
  </ul>
</div>
  );
}
```

Znany efekt

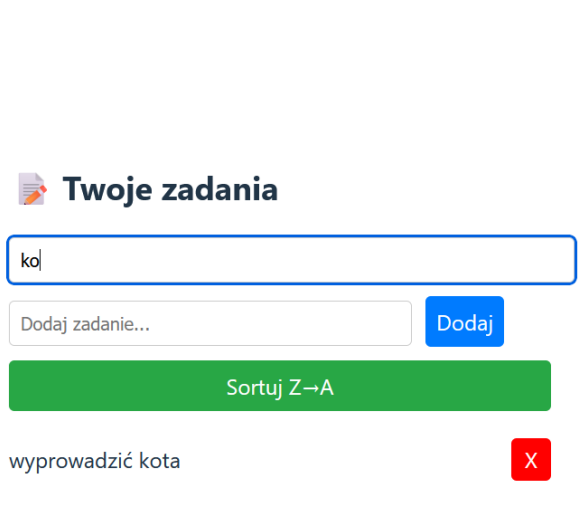

Vite + React
localhost:63294
Twoje zadania:
Dodaj zadanie...
Dodaj
pranie
X
sprzątanie
X
gotowanie
X

Wyszukiwarka



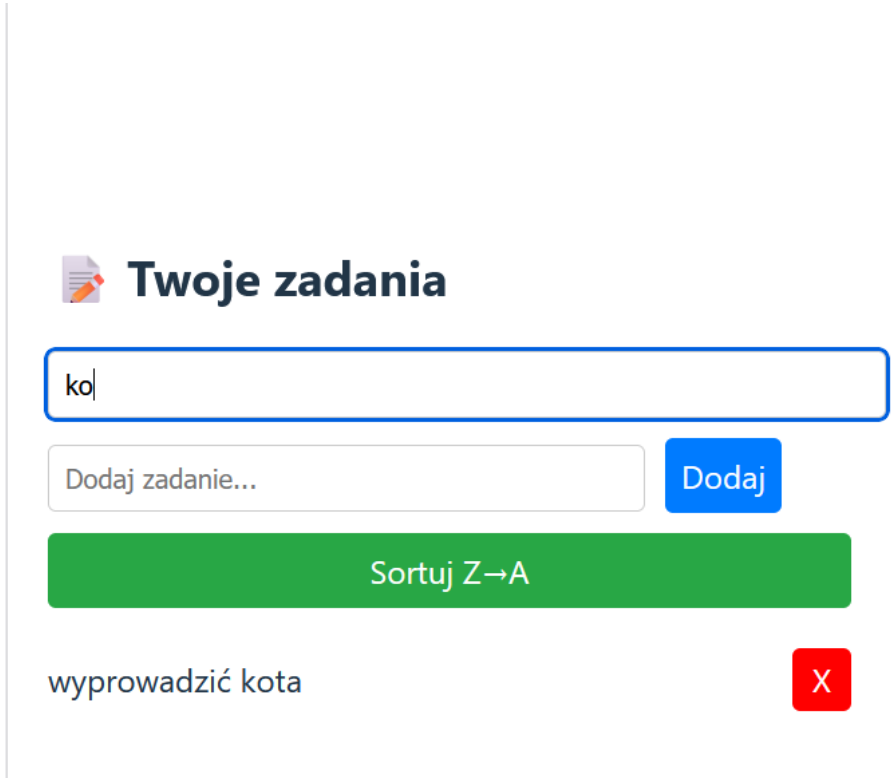
KOD PROJEKTU

```
import { useState, useEffect } from "react";

export default function App() {
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState("");
  const [searchTerm, setSearchTerm] = useState("");
  const [sortAsc, setSortAsc] = useState(true);

  useEffect(() => {
    const savedTasks = localStorage.getItem("tasks");
    if (savedTasks) {
      setTasks(JSON.parse(savedTasks));
    }
  }, []);

  useEffect(() => {
    localStorage.setItem("tasks", JSON.stringify(tasks));
  }, [tasks]);

  function addTask() {
    if (newTask.trim() === "") return;
    setTasks([...tasks, { text: newTask, completed: false }]);
    setNewTask("");
  }

  function deleteTask(index) {
    setTasks(tasks.filter((_, i) => i !== index));
  }

  function toggleTask(index) {
    const updatedTasks = tasks.map((task, i) =>
      i === index ? { ...task, completed: !task.completed } : task
    );
    setTasks(updatedTasks);
  }

  function sortTasks() {
    const sorted = [...tasks].sort((a, b) => {
      if (a.text.toLowerCase() < b.text.toLowerCase()) return sortAsc ? -1 : 1;
      if (a.text.toLowerCase() > b.text.toLowerCase()) return sortAsc ? 1 : -1;
      return 0;
    });
    setTasks(sorted);
    setSortAsc(!sortAsc);
  }

  const filteredTasks = tasks.filter(task =>
    task.text.toLowerCase().includes(searchTerm.toLowerCase())
  );

  return (
    <div style={{ padding: "20px", maxWidth: "400px", margin: "auto" }}>
      <h2>📝 Twoje zadania</h2>

      {/* 🔍 Pole wyszukiwania */}
      <input
        type="text"
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        placeholder="Szukaj zadania..."
        style={{
          width: "100%",
          padding: "8px",
          marginBottom: "10px",
          borderRadius: "4px",
          border: "1px solid #ccc",
        }}
      />

      {/* ➕ Pole dodawania nowego zadania */}
      <input
        type="text"
        value={newTask}
        onChange={(e) => setNewTask(e.target.value)}
        placeholder="Dodaj zadanie..."
        style={{
          width: "70%",
          padding: "8px",
          borderRadius: "4px",
          border: "1px solid #ccc",
        }}
      />
      <button
        onClick={addTask}
        style={{
          padding: "8px",
          marginLeft: "10px",
          backgroundColor: "#007bff",
          color: "white",
          border: "none",
          borderRadius: "4px",
          cursor: "pointer",
        }}
      >
        Dodaj
      </button>
```

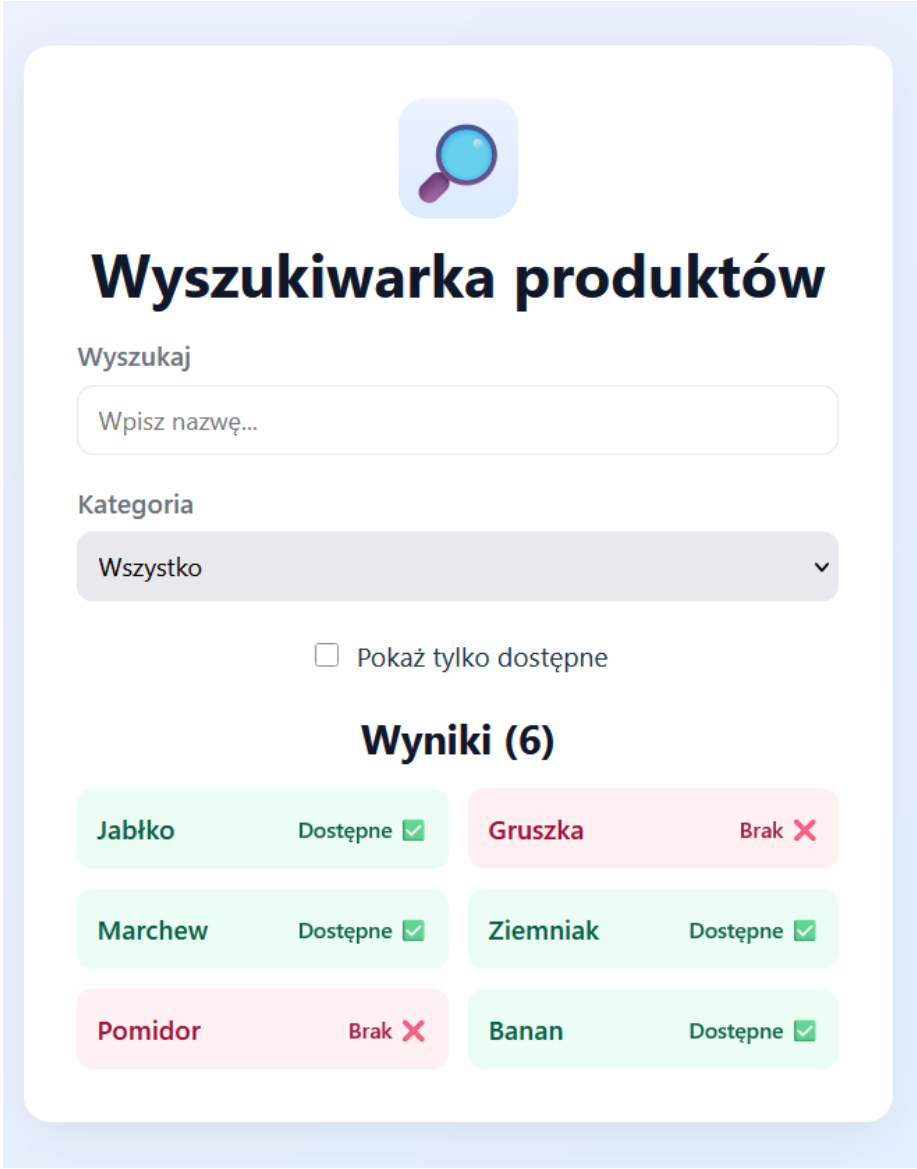


KOD PROJEKTU

```
<button
          onClick={sortTasks}
          style={{
            padding: "8px",
            marginTop: "10px",
            width: "100%",
            backgroundColor: "#28a745",
            color: "white",
            border: "none",
            borderRadius: "4px",
            cursor: "pointer",
          }}
        >
          Sortuj {sortAsc ? "A→Z" : "Z→A"}
        </button>
        {/* 📋 Lista zadań */}
        <ul style={{ listStyle: "none", padding: 0, marginTop: "20px" }}>
          {filteredTasks.length > 0 ? (
            filteredTasks.map((task, index) => (
              <li
                key={index}
                style={{
                  marginBottom: "10px",
                  display: "flex",
                  justifyContent: "space-between",
                  alignItems: "center",
                }}
              >
                <span
                  onClick={() => toggleTask(index)}
                  style={{
                    textDecoration: task.completed ? "line-through" : "none",
                    cursor: "pointer",
                  }}
                >
                  {task.text}
                </span>
                <button
                  onClick={() => deleteTask(index)}
                  style={{
                    background: "red",
                    color: "white",
                    border: "none",
                    padding: "5px 10px",
                    borderRadius: "4px",
                    cursor: "pointer",
                  }}
                >
                  X
                </button>
              </li>
            ))
          ) : (
            <p>Brak zadań do wyświetlenia.</p>
          )}
        </ul>
      </div>
    );
```



Wyszukiwarka z dodatkowymi opcjami



Kod aplikacji

```
import { useState } from "react";

export default function ProductSearch() {

  const products = [

    { name: "Jabłko", category: "Owoce", available: true },

    { name: "Gruszka", category: "Owoce", available: false },

    { name: "Marchew", category: "Warzywa", available: true },

    { name: "Ziemniak", category: "Warzywa", available: true },

    { name: "Pomidor", category: "Warzywa", available: false },

    { name: "Banan", category: "Owoce", available: true },

  ];

  const [search, setSearch] = useState("");

  const [category, setCategory] = useState("Wszystko");

  const [onlyAvailable, setOnlyAvailable] = useState(false);

  const filtered = products.filter((p) => {

    const matchName =
p.name.toLowerCase().includes(search.toLowerCase());

    const matchCategory = category === "Wszystko" || p.category
=== category;

    const matchAvailable = !onlyAvailable || p.available;

    return matchName && matchCategory && matchAvailable;

  });
```

```
return (
    <>
      {/* Wewnętrzny CSS — działa bez Tailwinda */}
      <style>{`
    :root{
      --card-bg: #ffffff;
      --page-bg: linear-gradient(135deg,#eef2ff 0%, #e6f0ff 100%);
      --accent: #4f46e5;
      --muted: #6b7280;
      --success: #ecfdf5;
      --danger: #fff1f2;
    }
    *{box-sizing: border-box; font-family: Inter, system-ui, -apple-system, "Segoe UI", Roboto, "Helvetica Neue", Arial;}
    body, #root { height: 100%; margin:0; }
    .page {
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
      padding: 24px;
      background: var(--page-bg);
    }
    .card {
      width: 100%;
      max-width: 920px;
      background: var(--card-bg);
      border-radius: 16px;
      box-shadow: 0 8px 30px rgba(20, 20, 40, 0.08);
      padding: 32px;
      text-align: center;
    }
    .icon {
      font-size: 48px;
      display: inline-flex;
      align-items:center;
      justify-content:center;
      width:72px;
      height:72px;
      border-radius: 16px;
      background: linear-gradient(180deg,#eef2ff,#dbeafe);
      margin-bottom: 12px;
    }
    h1 {
      margin: 0 0 18px 0;
      font-size: 36px;
      color: #0f172a;
    }

    .controls {
      display:flex;
      flex-wrap:wrap;
      gap:16px;
      justify-content:center;
      margin-bottom: 20px;
    }

    .control {
      min-width: 220px;
      flex: 1 1 260px;
      text-align: left;
    }

    .control label {
      display:block;
      margin-bottom:6px;
      color: var(--muted);
      font-weight:600;
    }

    .control input[type="text"],
    .control select {
      width:100%;
      padding:10px 12px;
      border:1px solid #e6e9ef;
      border-radius:10px;
      font-size:15px;
      outline:none;
      transition: box-shadow .15s, border-color .15s;
    }
    .control input[type="text"]:focus,
    .control select:focus {
      box-shadow: 0 4px 18px rgba(79,70,229,0.12);
      border-color: rgba(79,70,229,0.6);
    }

    .checkbox-row {
      display:flex;
      align-items:center;
      gap:8px;
      justify-content:center;
      margin-bottom: 12px;
    }

    .results {
      margin-top: 12px;
      text-align:center;
    }

    .results h2 {
      margin-bottom: 12px;
      color: #0f172a;
    }

    .list {
      display:grid;
      grid-template-columns: repeat(auto-fit,minmax(200px,1fr));
      gap:12px;
    }

    .item {
      padding:12px;
      border-radius:10px;
      display:flex;
      justify-content:space-between;
      align-items:center;
      gap:10px;
      font-weight:600;
    }

    .item.available { background: #ecfdf5; color:#065f46; box-shadow: inset 0 1px 0 rgba(255,255,255,0.6); }
    .item.unavailable { background: #fff1f2; color:#9f1239; }

    /* responsywność */
    @media (max-width:600px){
      .control { flex-basis:100%; }
      h1 { font-size:28px; }
    }
  `}</style>
```

Kod aplikacji

Alternatywą jest użycie Bootstrap lub Tailwind. O nich w dalszej części.

Wewnętrzny styl CSS

Kod aplikacji

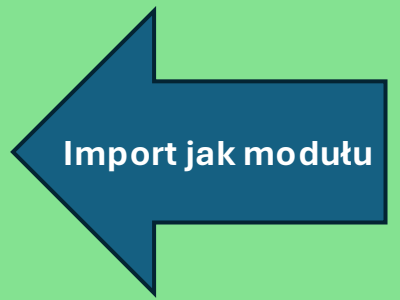
```
<div className="page">
        <div className="card" role="region" aria-label="Wyszukiwarka produktów">
          <div className="icon">🔎</div>
          <h1>Wyszukiwarka produktów</h1>

          <div className="controls" aria-hidden={false}>
            <div className="control">
              <label htmlFor="search">Wyszukaj</label>
              <input
                id="search"
                type="text"
                placeholder="Wpisz nazwę..."
                value={search}
                onChange={(e) => setSearch(e.target.value)}
              />
            </div>

            <div className="control">
              <label htmlFor="category">Kategoria</label>
              <select id="category" value={category} onChange={(e) => setCategory(e.target.value)}>
                <option>Wszystko</option>
                <option>Owoce</option>
                <option>Warzywa</option>
              </select>
            </div>
          </div>

          <div className="checkbox-row">
            <input
              id="onlyAvail"
              type="checkbox"
              checked={onlyAvailable}
              onChange={(e) => setOnlyAvailable(e.target.checked)}
            />
            <label htmlFor="onlyAvail">Pokaż tylko dostępne</label>
          </div>

          <div className="results">
            <h2>Wyniki ({filtered.length})</h2>
            {filtered.length === 0 ? (
              <p style={{ color: "#6b7280", fontStyle: "italic" }}>Brak wyników</p>
            ) : (
              <ul className="list" style={{ listStyle: "none", padding: 0, margin: 0 }}>
                {filtered.map((item, i) => (
                  <li
                    key={i}
                    className={`item ${item.available ? "available" : "unavailable"}`}
                    aria-label={`${item.name} - ${item.available ? "Dostępne" : "Brak"}`}
                  >
                    <span>{item.name}</span>
                    <span style={{ fontSize: 13, opacity: 0.95 }}>
                      {item.available ? "Dostępne ✅" : "Brak ❌"}
                    </span>
                  </li>
                ))}
              </ul>
            )}
          </div>
        </div>
      </div>
    </>
  );
}
```

Najważniejsza część kodu

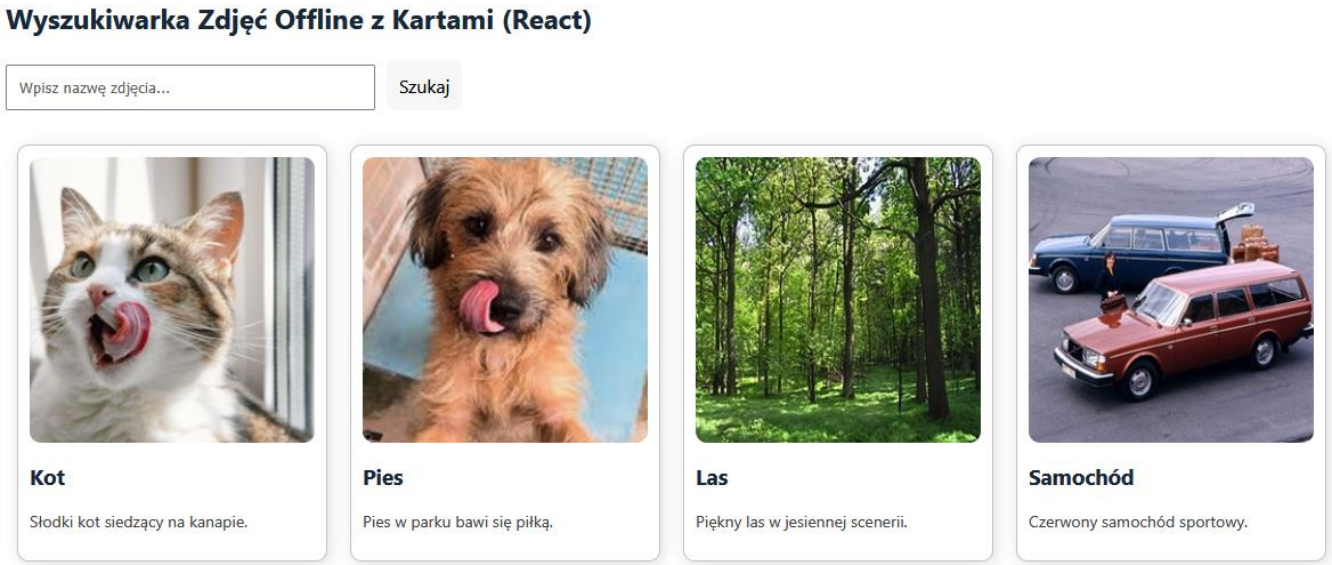
Assets

```
vite-react-obrazy/
├── public/
├── src/
│   ├── assets/
│   │   ├── logo.png
│   │   └── kot.jpg
│   ├── App.jsx
│   ├── main.jsx
└── index.html
```

```jsx
import logo from "./assets/logo.png";
import kot from "./assets/kot.jpg";

function App() {
 return (
  <div style={styles.container}>
   <h1>Obsługa zdjęć w React Vite </h1>

   {/* Importowany obraz z assets */}
   <div style={styles.section}>
    <h3>Logo (importowane):</h3>
    <img src={logo} alt="Logo" style={styles.image} />
   </div>

   {/* Importowany drugi obraz */}
   <div style={styles.section}>
    <h3>Kot (importowany):</h3>
    <img src={kot} alt="Kot" style={styles.image} />
   </div>

   {/* Obraz z public (gdybyś chciał) */}
   <div style={styles.section}>
    <h3>Obraz z folderu public:</h3>
    <img src="/pies.jpg" alt="Pies" style={styles.image} />
    <p>
     (plik <code>pies.jpg</code> musi być w folderze
<code>/public</code>)
    </p>
   </div>
  </div>
 );
}

const styles = {
 container: {
  fontFamily: "Arial, sans-serif",
  padding: "20px",
  textAlign: "center",
 },
 section: {
  margin: "20px 0",
 },
 image: {
  maxWidth: "200px",
  borderRadius: "8px",
  boxShadow: "0 2px 6px rgba(0,0,0,0.2)",
 },
};

export default App;
```

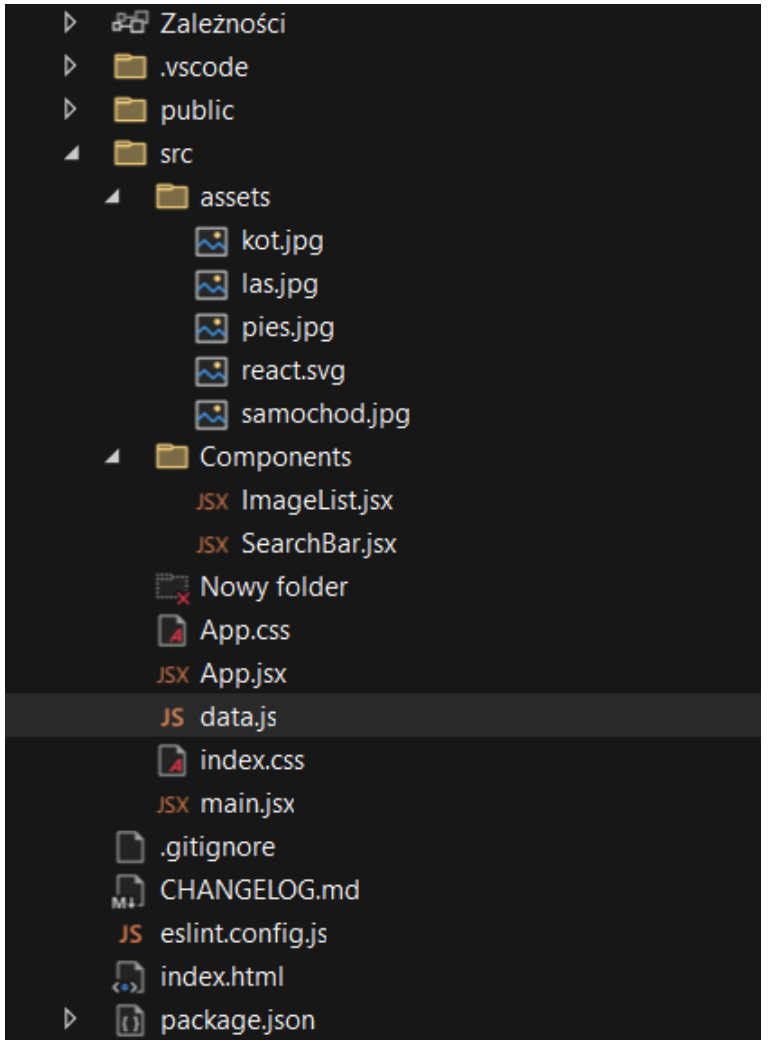


Wyszukiwarka zdjęć (z kartami)

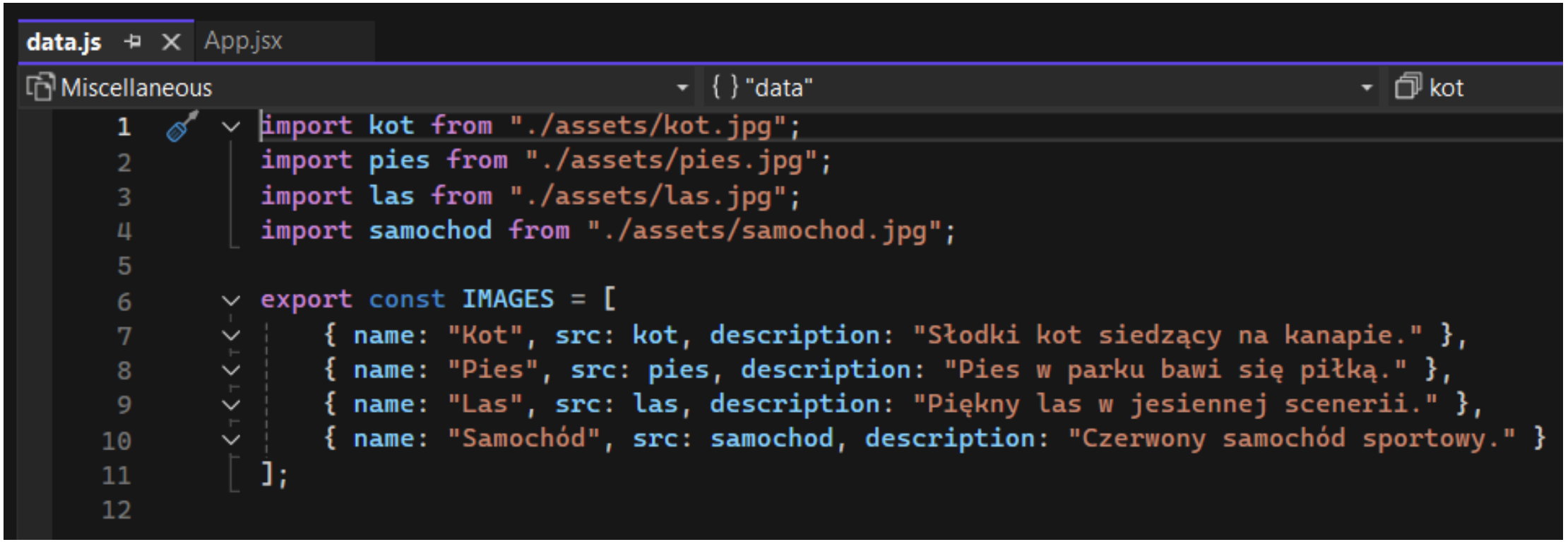


Wyszukiwarka zdjęć (z kartami)

Struktura projektu



Wyszukiwarka zdjęć (z kartami)



Komponenty

```
import React, { useState } from "react";

function SearchBar({ onSubmit }) {
  const [term, setTerm] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    onSubmit(term);
  };

  return (
    <div style={{ marginBottom: "20px" }}>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Wpisz nazwę zdjęcia..."
          value={term}
          onChange={(e) => setTerm(e.target.value)}
          style={{ width: "300px", padding: "10px" }}
        />
        <button type="submit" style={{ padding: "10px", marginLeft: "10px" }}>
          Szukaj
        </button>
      </form>
    </div>
  );
}

export default SearchBar;
```

Komponenty

```
import React from "react";


function ImageList({ images }) {
  const imgElements = images.map((img) => {
    return (
      <div
        key={img.name}
        style={{
          border: "1px solid #ccc",
          borderRadius: "10px",
          margin: "10px",
          width: "250px",
          padding: "10px",
          boxShadow: "2px 2px 12px rgba(0,0,0,0.1)"
        }}
      >
        <img
          src={img.src}
          alt={img.name}
          style={{ width: "100%", borderRadius: "10px" }}
        />
        <h3 style={{ margin: "10px 0 5px 0" }}>{img.name}</h3>
        <p style={{ fontSize: "14px", color: "#555"
}}>{img.description}</p>
      </div>
    );
  });

  return (
    <div style={{ display: "flex", flexWrap: "wrap" }}>
      {imgElements}
    </div>
  );
}

export default ImageList;
```

APP.JSX – główna funkcja wyszukiwarki

```
import React, { useState } from "react";
import SearchBar from "./components/SearchBar";
import ImageList from "./components/ImageList";
import { IMAGES } from "./data";

function App() {
  const [images, setImages] = useState(IMAGES);

  const onSearchSubmit = (term) => {
    const filtered = IMAGES.filter((img) =>
      img.name.toLowerCase().includes(term.toLowerCase())
    );
    setImages(filtered);
  };

  return (
    <div style={{ margin: "20px" }}>
      <h2>Wyszukiwarka Zdjęć Offline z Kartami (React)</h2>
      <SearchBar onSubmit={onSearchSubmit} />
      <ImageList images={images} />
    </div>
  );
}

export default App;
```

Podsumowanie - pytania

Jak nazywa się menedżer pakietów JavaScript?

Jaki numer ma standardowy port w React?

Jakiego rodzaju pliki znajdują się w strukturze typowego projektu?

Do czego służy handler?

Jak działa klasa Tasks i jakie ma zastosowanie?

Do czego służy SET w React?

Do czego służy mapowanie list?

Zadania do samodzielnego wykonania

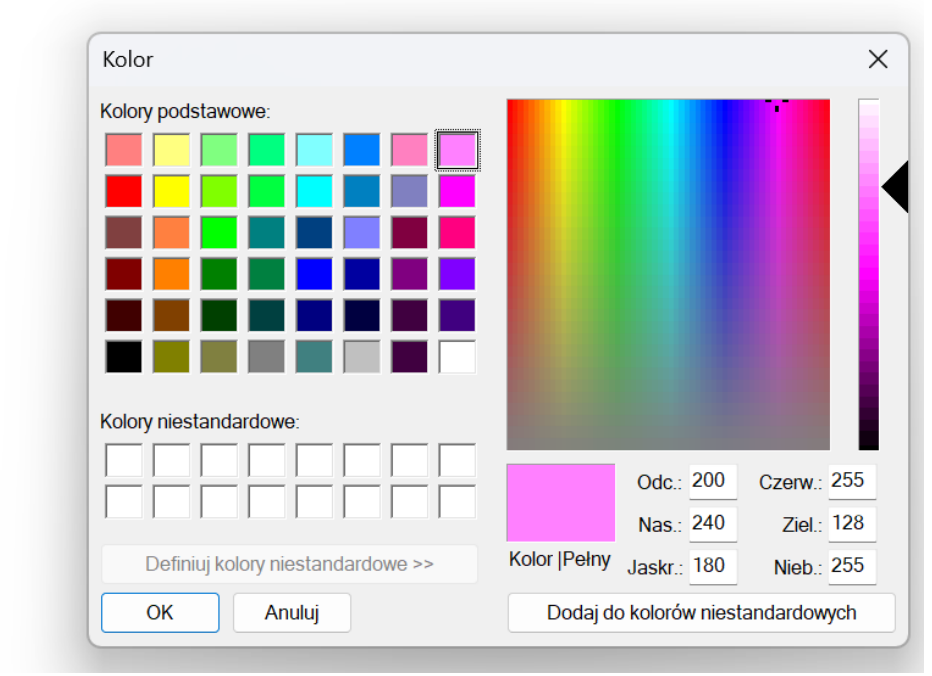
Zadanie 1

Wykonaj prosty zegar odliczający sekundy.

Zadania do samodzielnego wykonania

Zadanie 2

Po naciśnięciu pola typu kolor tło zmienia kolor i wyświetla się jego numer.



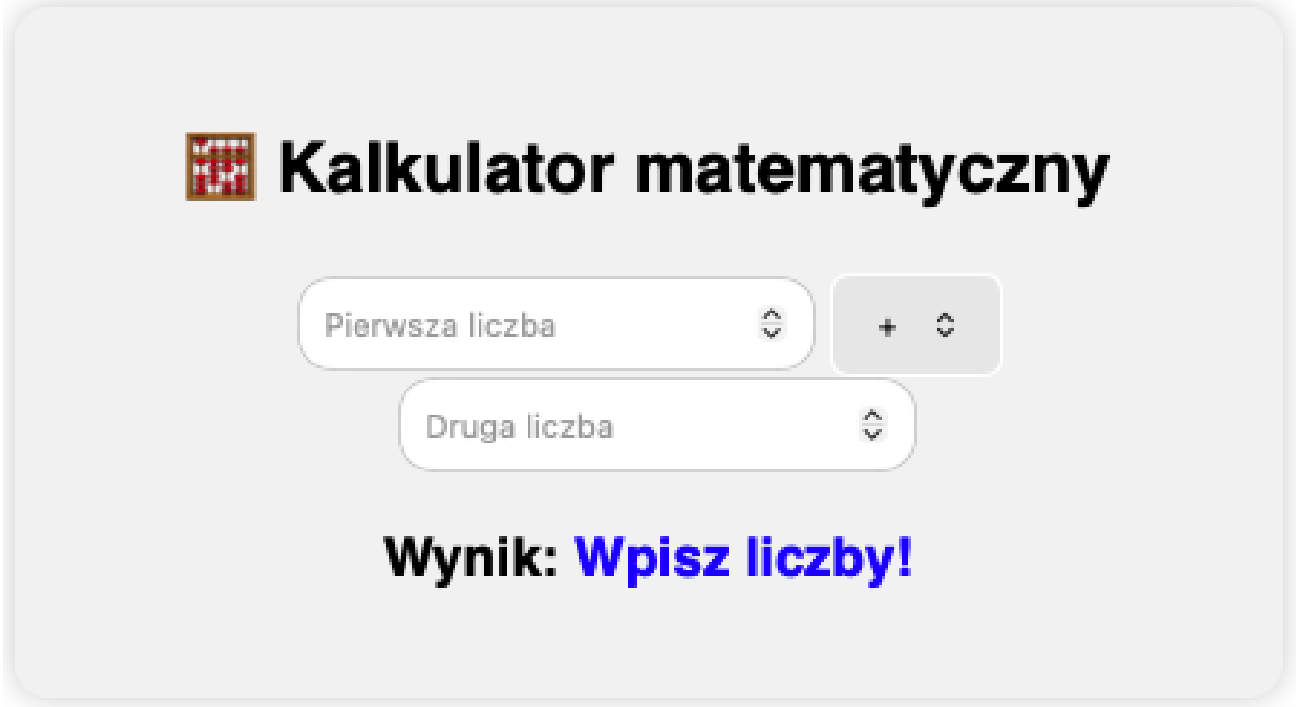
Zadania do samodzielnego wykonania

Zadanie 3

Wykonaj kalkulator według wzoru zamieszczonego obok.

Użytkownik wprowadza dwie liczby i ma możliwość wyboru działania.

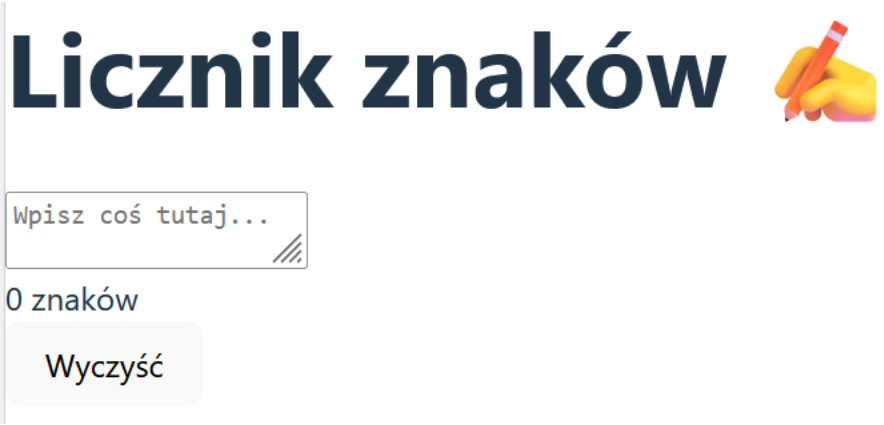
Do wyboru jest: dodawanie, odejmowanie, mnożenie, dzielenie i potęgowanie. Wynik pokazuje się obok etykiety.



Zadania do samodzielnego wykonania

Zadanie 4

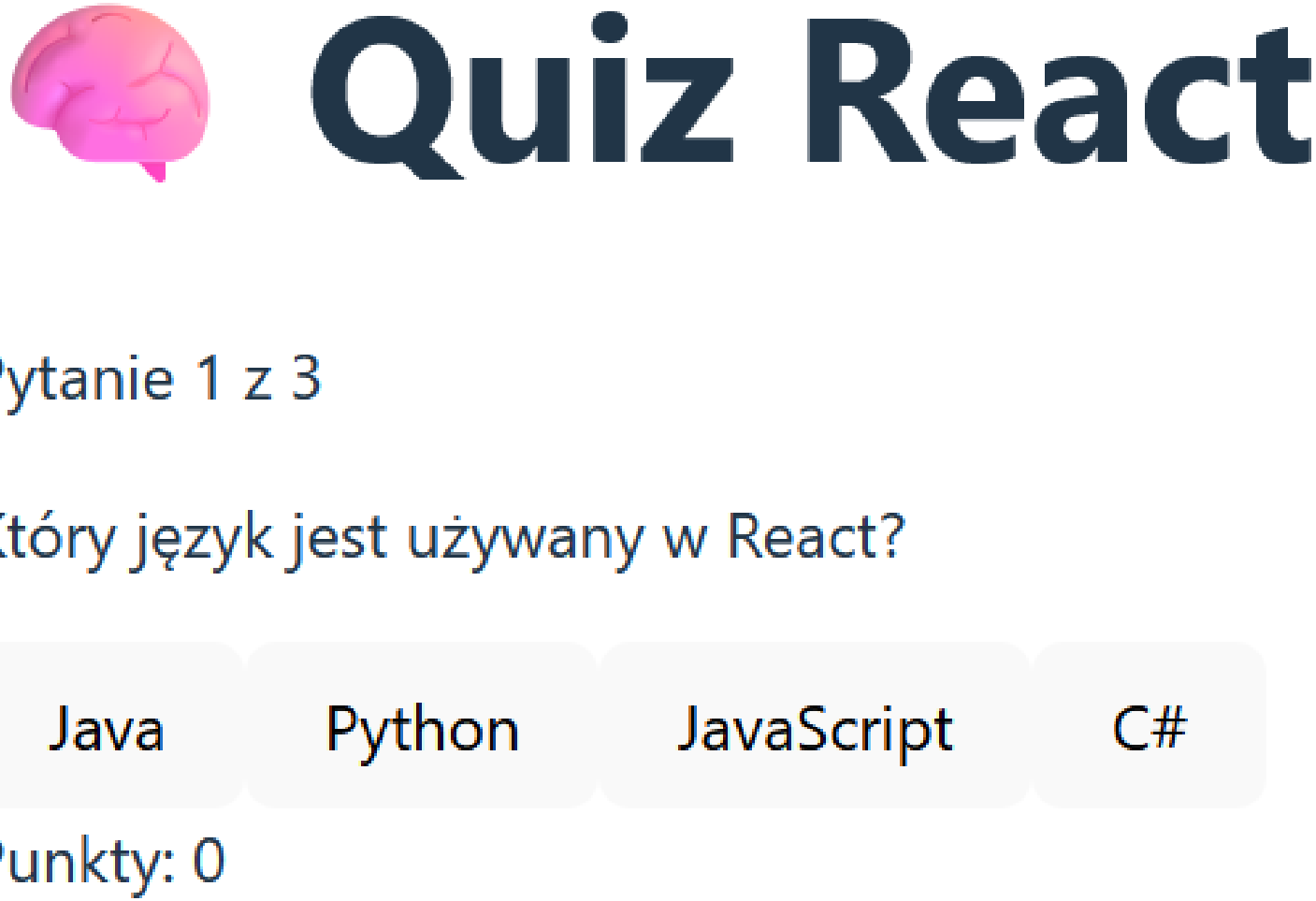
Napisz prostą aplikację, której zadaniem jest zliczanie znaków wpisanych do pola tekstowego. Do aplikacji dodaj przycisk do czyszczenia pola.



Zadania do samodzielnego wykonania

Zadanie 5 (na 5)

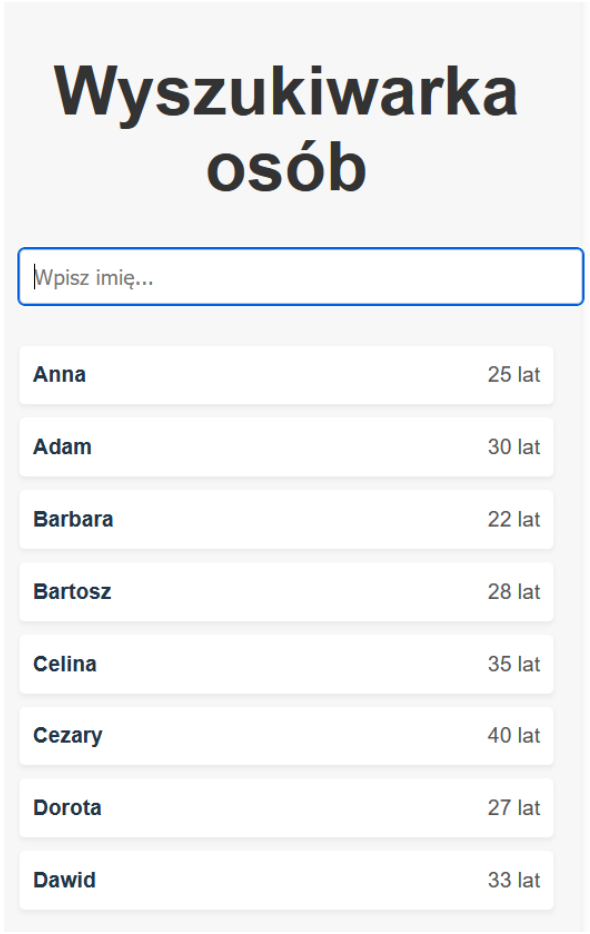
Napisz krótki test o React. Aplikacja wykorzystuje tablice do przechowywania danych oraz gromadzi punkty za poprawne odpowiedzi.

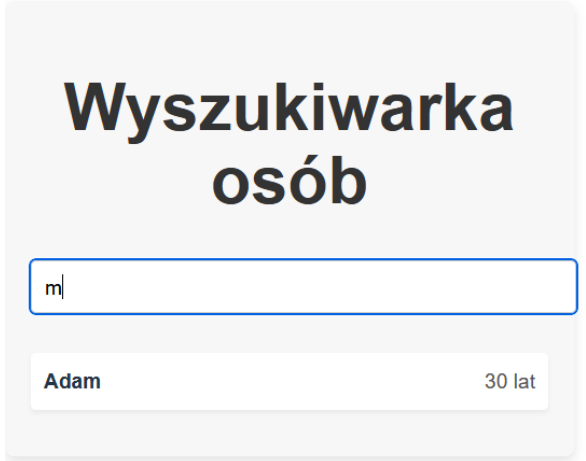


Zadania do samodzielnego wykonania

Zadanie 6

Utwórz wyszukiwarkę danych osobowych według poniższego wzoru. Dane są pobierane z tablicy. Aplikacja wykorzystuje karty do wyświetlania danych.



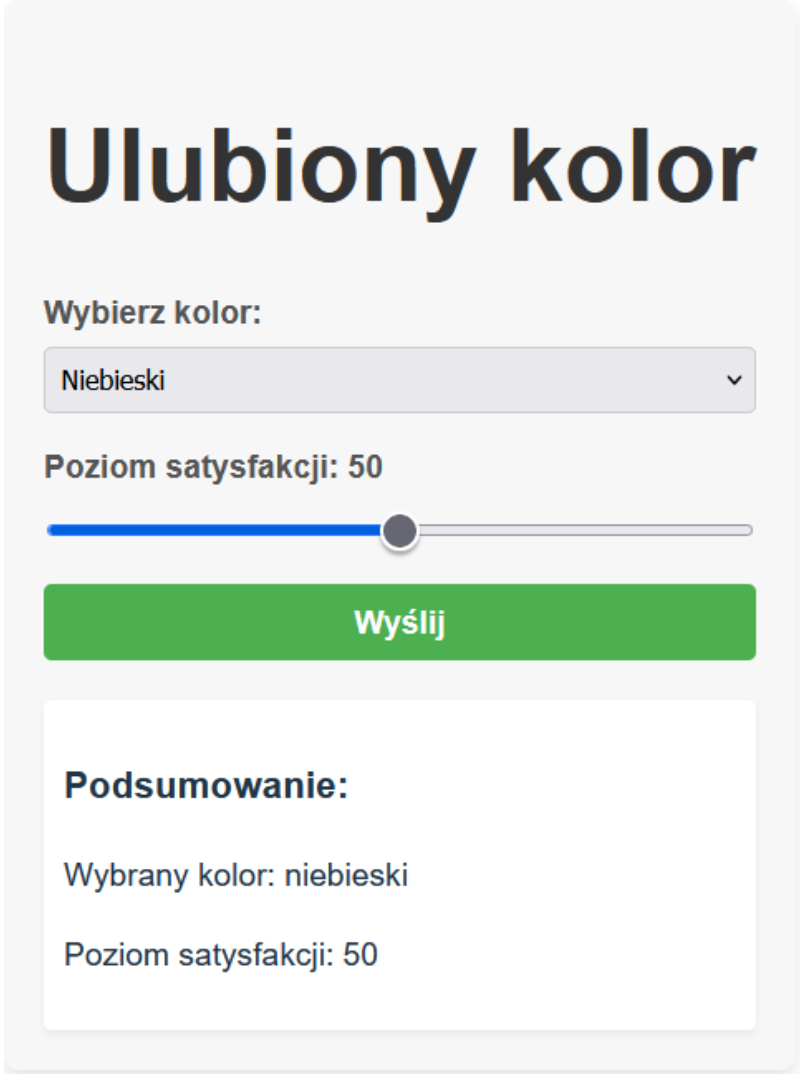


Zadania do samodzielnego wykonania

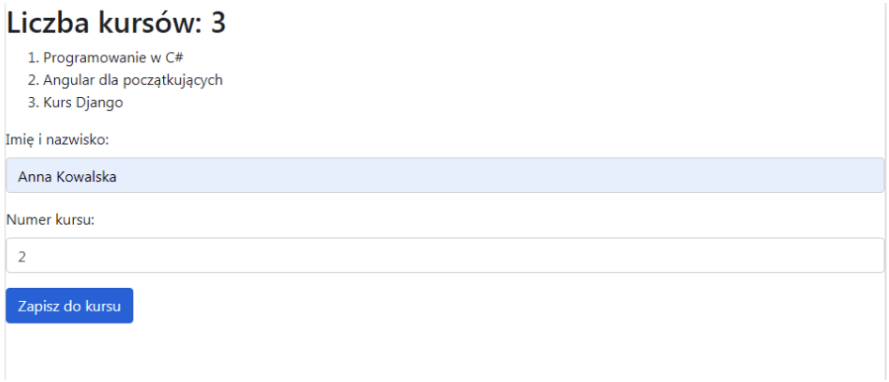
Zadanie 7

Aplikacja daje możliwość pobrania ulubionego koloru oraz poziomu satysfakcji z wyboru w zakresie <0,100>.

Dane są wyświetlane pod przyciskiem.

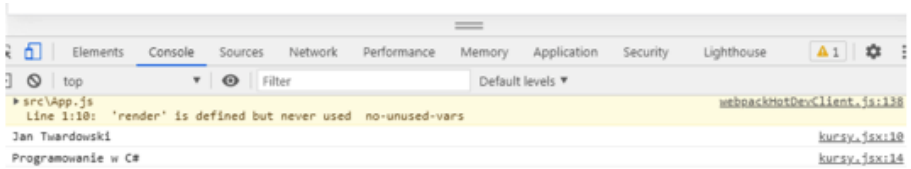


Zadania do samodzielnego wykonania



Zadanie 8

Aplikacja pozwala zapisać użytkownika do określonego kursu. Po naciśnięciu przycisku zapisane dane wyświetlają się w konsoli.



Zadania do samodzielnego wykonania

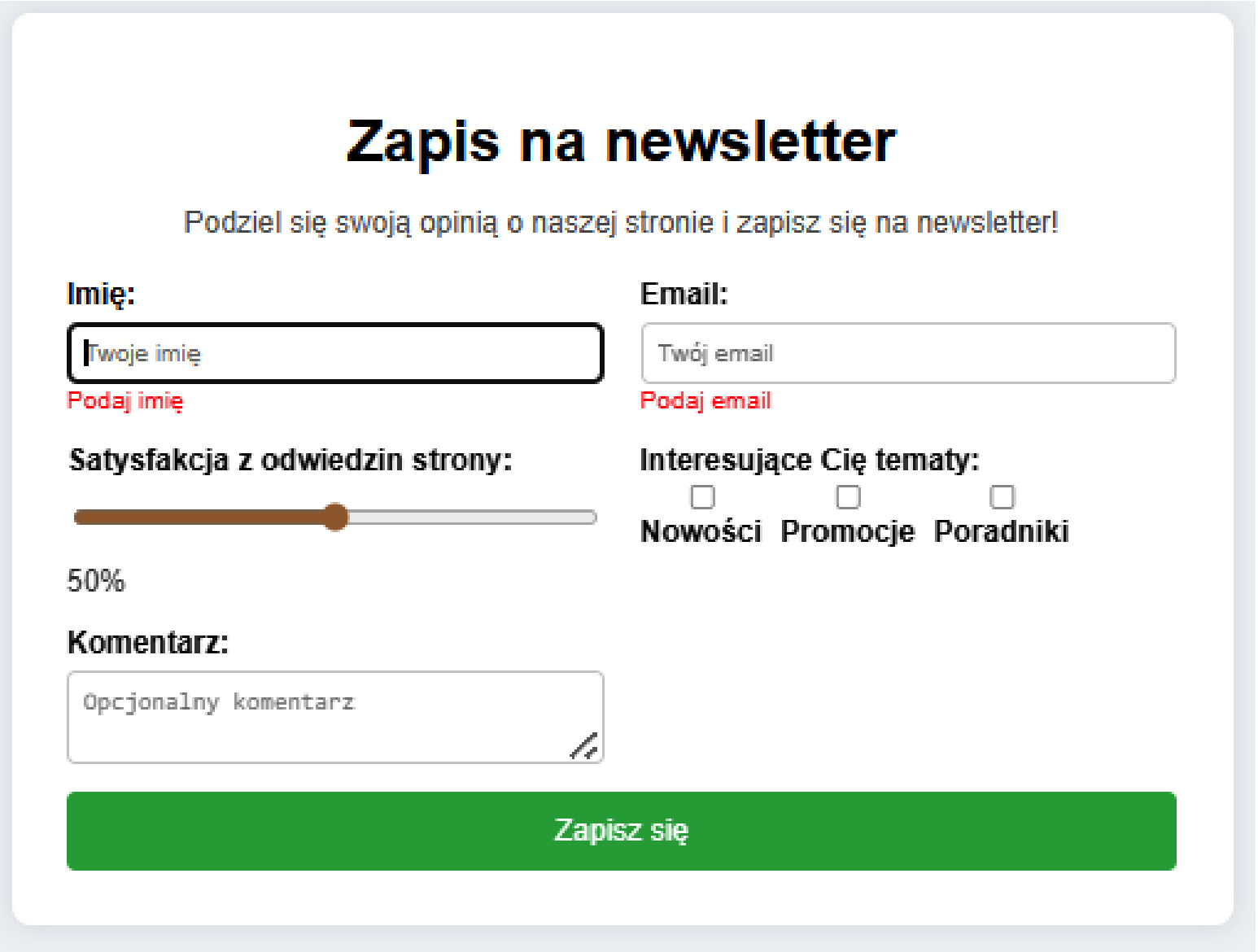
Zadanie 9

W aplikacji klient może wybrać kolor koszulki, rozmiar oraz wczytać plik ze zdjęciem do wydruku. Po dokonaniu wyboru wyświetlają się wybrane opcje oraz zdjęcie.

Zadania do samodzielnego wykonania

Zadanie 10

Formularz zbiera dane od klienta i przekazuje je do okna typu <<alert>>. Korzysta z walidacji za pomocą React Hook Form.

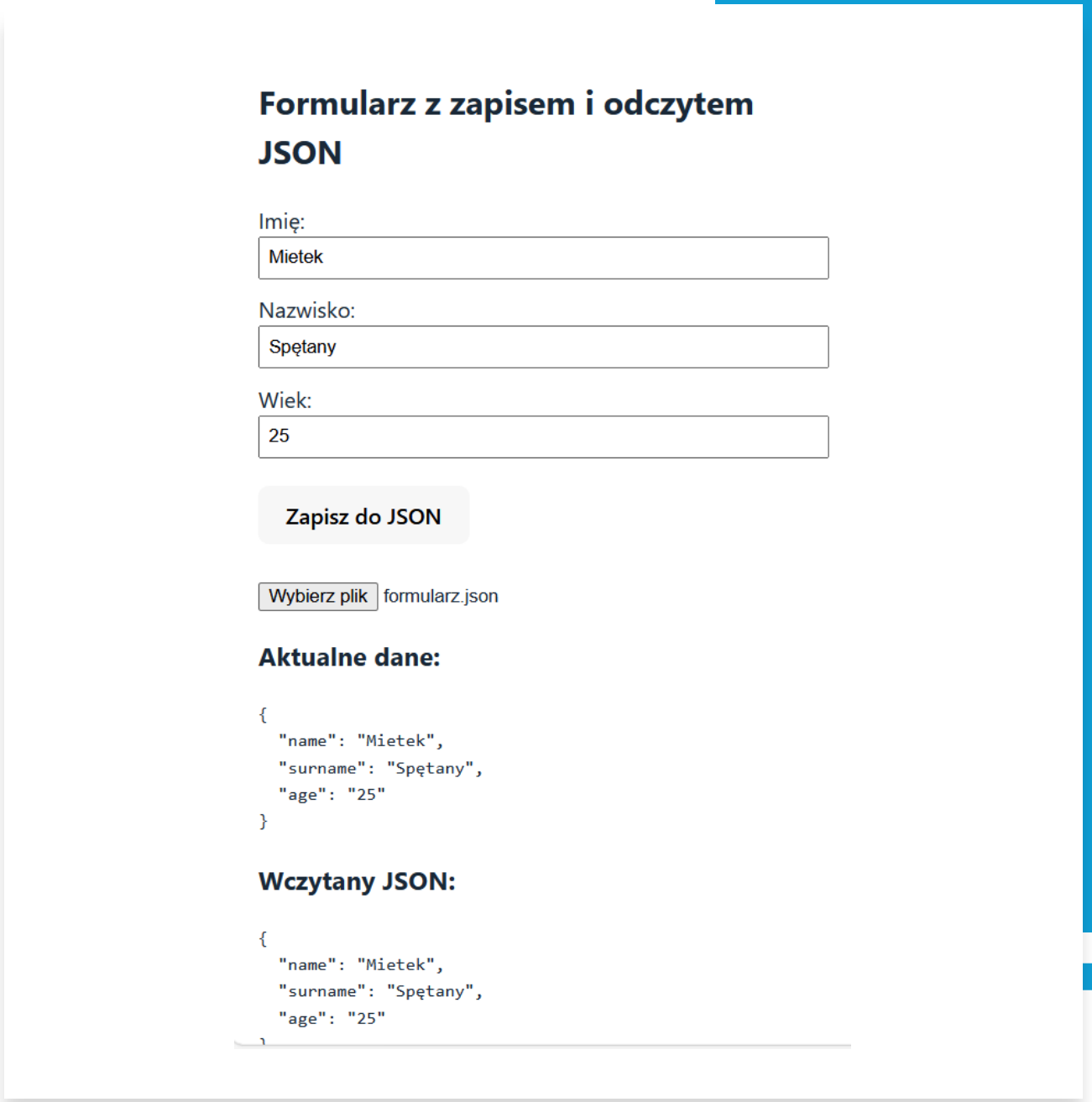


Zadania do samodzielnego wykonania

Zadanie 11

Wykonaj wyszukiwarkę zdjęć, która działa lokalnie. Wyszukiwanie działa po rozpoczęciu wpisywania. Po wyszukaniu na ekranie widać jedynie wyszukany element.

Zapisywanie i odczytywanie danych w pliku

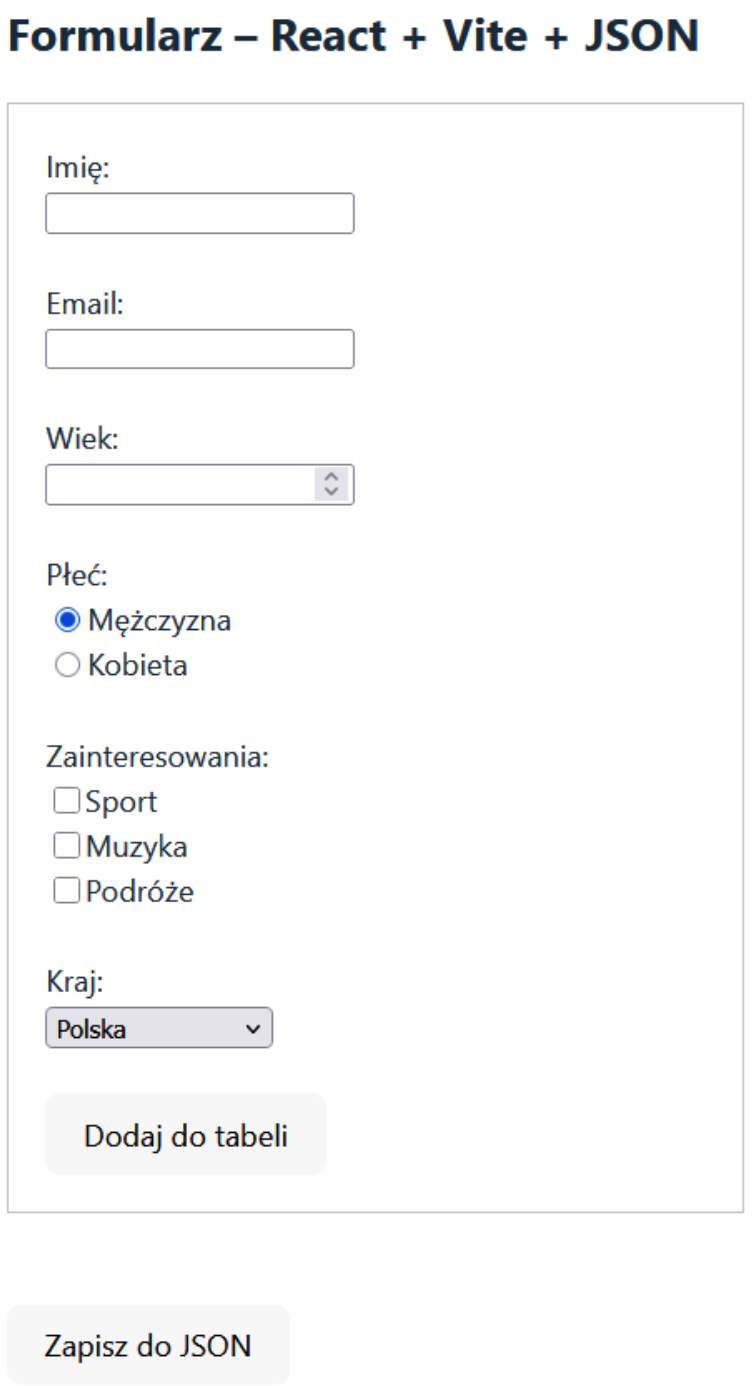


Zapis i odczyt – plik JSON

```
import { useState } from "react";

export default function FormJsonApp() {

 const [form, setForm] = useState({

  name: "",

  surname: "",

  age: ""

 });

 const [loadedData, setLoadedData] = useState(null);

 // obsługa zmian w formularzu

 const handleChange = (e) => {

  const { name, value } = e.target;

  setForm((prev) => ({ ...prev, [name]: value }));

 };

 // zapis do pliku JSON

 const saveJson = () => {

  const json = JSON.stringify(form, null, 2);

  const blob = new Blob([json], { type: "application/json" });

  const url = URL.createObjectURL(blob);

  const a = document.createElement("a");

  a.href = url;

  a.download = "formularz.json";

  a.click();

  URL.revokeObjectURL(url);

 };


 // odczyt pliku JSON

 const loadJson = (event) => {

  const file = event.target.files[0];


  if (!file) return;

  const reader = new FileReader();

  reader.onload = (e) => {

   try {

    const json = JSON.parse(e.target.result);

    setLoadedData(json);

    setForm(json); // ← automatyczne wstawienie danych do formularza

   } catch (err) {

    alert("Błędny format JSON");

   }

  };

  reader.readAsText(file);

 };

 return (

  <div style={{ padding: 20, maxWidth: "400px" }}>

   <h2>Formularz z zapisem i odczytem JSON</h2>

   <label>Imię:</label>

   <input

    type="text"

    name="name"

    value={form.name}

    onChange={handleChange}

    style={{ width: "100%", padding: 6, marginBottom: 10 }}

   />

   <label>Nazwisko:</label>

   <input

    type="text"

    name="surname"

    value={form.surname}

    onChange={handleChange}

    style={{ width: "100%", padding: 6, marginBottom: 10 }}

   />

   <label>Wiek:</label>

   <input

    type="number"

    name="age"

    value={form.age}

    onChange={handleChange}


    style={{ width: "100%", padding: 6, marginBottom: 20 }}

   />

   <button onClick={saveJson}>Zapisz do JSON</button>

   <br /><br />

   <input type="file" accept=".json" onChange={loadJson} />

   <h3>Aktualne dane:</h3>

   <pre>{JSON.stringify(form, null, 2)}</pre>


   {loadedData && (

    <>

     <h3>Wczytany JSON:</h3>

<pre>{JSON.stringify(loadedData, null, 2)}</pre>

    </>

   )}

  </div>

 );

}
```

Zapis i odczyt – plik JSON

Aplikacja pobiera z formularza dane a następnie pozwala je zapisać i wyświetlić. Plik JSON zawiera dane sortowane na zasadzie „klucz-wartość”. Dane możemy zapisywać i wyświetlać w postaci listy lub tabeli. Oczywiście możemy również je sortować i zmieniać – podobnie jak ma to w przypadku wykorzystania języka PHP i bazy danych SQL.



Zapis i odczyt – plik JSON

Zapisz do JSON

Przeglądaj... Nie wybrano pliku.

Tabela danych:

Imię	Email	Wiek	Płeć	Zainteresowania	Kraj
Stefan Batory	batory@wp.pl	34	male	travel	Polska

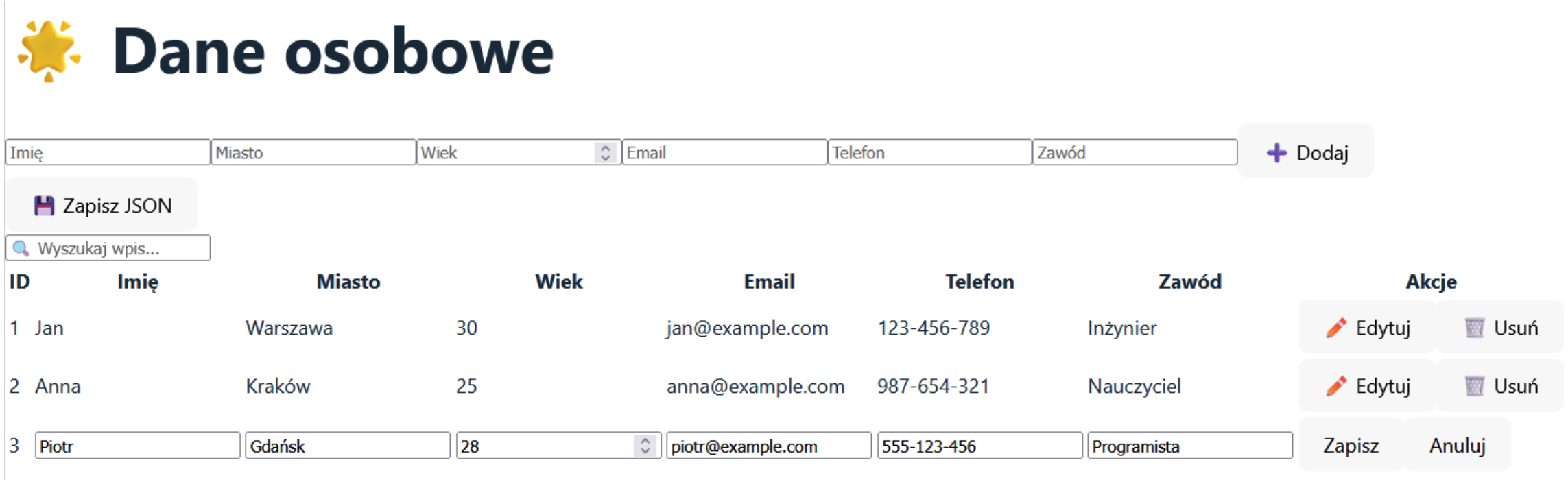
ZAPIS I ODCZYT PLIK JSON

```
import React, { useState } from "react";


export default function FormExample() {
  const [form, setForm] = useState({
    name: "",
    email: "",
    age: "",
    gender: "male",
    interests: {
      sport: false,
      music: false,
      travel: false,
    },
    country: "Polska",
  });
  const [tableData, setTableData] = useState([]);
  // --- Obsługa pól tekstowych ---
  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };
  // --- Checkboxy (zagnieżdżone dane) ---
  const handleCheckbox = (e) => {
    setForm({
      ...form,
      interests: {
        ...form.interests,
        [e.target.name]: e.target.checked,
      },
    });
  };
  // --- Zapis formularza do tabeli ---
  const addToTable = () => {
    setTableData([...tableData, form]);
  };


  // --- ZAPIS DO JSON ---
  const saveToJSON = () => {
    const blob = new Blob([JSON.stringify(tableData, null, 2)], {
      type: "application/json",
    });
    const url = URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = "dane.json";
    a.click();
    URL.revokeObjectURL(url);
  };

  // --- ODCZYT JSON ---
  const loadJSON = (event) => {
    const file = event.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (e) => {
      try {
        const json = JSON.parse(e.target.result);
        setTableData(json);
      } catch {
        alert("Błędny plik JSON!");
      }
    };
    reader.readAsText(file);
  };
  return (
    <div style={{ padding: 20 }}>
      <h2>Formularz – React + Vite + JSON</h2>
      {/* --- FORMULARZ --- */}
      <div style={{ border: "1px solid #ccc", padding: 20, width: 350 }}>
        <label>
          Imię:<br />
          <input
            name="name"
            value={form.name}
            onChange={handleChange}
            type="text"
          />
        </label>
        <br /><br />
        <label>
          Email:<br />
          <input
            name="email"
            value={form.email}
            onChange={handleChange}
            type="email"
          />
        </label>
        <br /><br />
        <label>
          Wiek:<br />
          <input
            name="age"
            value={form.age}
            onChange={handleChange}
            type="number"
          />
        </label>
        <br /><br />
        {/* --- RADIO --- */}
        <div>
          Płeć:<br />
          <label>
            <input
              type="radio"
              name="gender"
              value="male"
              checked={form.gender === "male"}
              onChange={handleChange}
            />
            Mężczyzna
          </label>
          <br />
          <label>
            <input
              type="radio"
              name="gender"
              value="female"
              checked={form.gender === "female"}
              onChange={handleChange}
            />
            Kobieta
          </label>
        </div>
        <br />
```

Zapis i odczyt – plik JSON

```
{/* --- CHECKBOXY --- */}
      <div>
       Zainteresowania:
       <br />
       <label>
        <input
         type="checkbox"
         name="sport"
         checked={form.interests.sport}
         onChange={handleCheckbox}
        />
        Sport
       </label>
       <br />
       <label>
        <input
         type="checkbox"
         name="music"
         checked={form.interests.music}
         onChange={handleCheckbox}
        />
        Muzyka
       </label>
       <br />
       <label>
        <input
         type="checkbox"
         name="travel"
         checked={form.interests.travel}
         onChange={handleCheckbox}
        />
        Podróże
       </label>
      </div>
      <br />
      {/* --- SELECT --- */}
      <label>
       Kraj:<br />
       <select name="country"
value={form.country}
onChange={handleChange}>
        <option>Polska</option>
        <option>Niemcy</option>
        <option>USA</option>
        <option>Wielka Brytania</option>
       </select>
      </label>

      <br /><br />
       <button onClick={addToTable}>Dodaj
do tabeli</button>
      </div>
      <br /><br />
      {/* --- ZAPIS I ODCZYT JSON --- */}
      <button onClick={saveToJSON}>Zapisz
do JSON</button>
      <br /><br />
      <input type="file" accept=".json"
onChange={loadJSON} />
      <br /><br />
      {/* --- TABELA DANYCH --- */}
      <h3>Tabela danych:</h3>
      <table border="1" cellPadding="5">
       <thead>
        <tr>
         <th>Imię</th>
         <th>Email</th>
         <th>Wiek</th>
         <th>Płeć</th>
         <th>Zainteresowania</th>
         <th>Kraj</th>
        </tr>
       </thead>
       <tbody>
{tableData.map((row, i) => (
 <tr key={i}>
  <td>{row.name}</td>
  <td>{row.email}</td>
  <td>{row.age}</td>
  <td>{row.gender}</td>
  <td>

   {Object.entries(row.interests)

    .filter(([k, v]) => v)
    .map(([k]) => k)
    .join(', ')}
  </td>
  <td>{row.country}</td>
 </tr>
))}
       </tbody>
      </table>
     </div>
    );
}
```

Zapis, wyszukiwanie i odczyt – kompletny przykład



Zapis, wyszukiwanie i odczyt – kompletny przykład

```
import React, { useState } from "react";

export default function SearchTable(){

  const [data, setData] = useState([

    { id: 1, name: "Jan", city: "Warszawa", age: 30, email: "jan@example.com", phone: "123-456-789", occupation: "Inżynier" },

    { id: 2, name: "Anna", city: "Kraków", age: 25, email: "anna@example.com", phone: "987-654-321", occupation: "Nauczyciel" },

    { id: 3, name: "Piotr", city: "Gdańsk", age: 28, email: "piotr@example.com", phone: "555-123-456", occupation: "Programista" },

    { id: 4, name: "Kasia", city: "Poznań", age: 32, email: "kasia@example.com", phone: "444-789-012", occupation: "Lekarz" }

  ]);

  const [form, setForm] = useState({ name: "", city: "", age: "", email: "", phone: "", occupation: "" });

  const [query, setQuery] = useState("");

  const [editingId, setEditingId] = useState(null);

  const [editForm, setEditForm] = useState({ name: "", city: "", age: "", email: "", phone: "", occupation: "" });

  const [nextId, setNextId] = useState(5); // Licznik dla unikalnych ID

  const handleFormChange = (e) => {

    setForm({ ...form, [e.target.name]: e.target.value });

  };

  const handleAdd = () => {

    if (!form.name || !form.city || !form.age || !form.email || !form.phone || !form.occupation) return;

    const newEntry = {

      id: nextId,

      name: form.name,

      city: form.city,

      age: parseInt(form.age),

      email: form.email,

      phone: form.phone,

      occupation: form.occupation

    };

    setData([...data, newEntry]);

    setForm({ name: "", city: "", age: "", email: "", phone: "", occupation: "" });

    setNextId(nextId + 1);

  };

  const handleEdit = (row) => {

    setEditingId(row.id);

    setEditForm({ name: row.name, city: row.city, age: row.age, email: row.email, phone: row.phone, occupation: row.occupation });

  };



  const handleSaveEdit = () => {

    if (!editForm.name || !editForm.city || !editForm.age || !editForm.email || !editForm.phone || !editForm.occupation) return;

    setData(data.map(row =>

      row.id === editingId

        ? { ...row, name: editForm.name, city: editForm.city, age: parseInt(editForm.age), email: editForm.email, phone: editForm.phone, occupation: editForm.occupation }

        : row

    ));

    setEditingId(null);

    setEditForm({ name: "", city: "", age: "", email: "", phone: "", occupation: "" });

  };

  const handleCancelEdit = () => {

    setEditingId(null);

    setEditForm({ name: "", city: "", age: "", email: "", phone: "", occupation: "" });

  };

  const handleDelete = (id) => {

    setData(data.filter(row => row.id !== id));

  };

  const handleEditFormChange = (e) => {

    setEditForm({ ...editForm, [e.target.name]: e.target.value });

  };

  const filtered = data.filter((row) =>

    Object.values(row)

      .join(" ")

      .toLowerCase()

      .includes(query.toLowerCase())

  );

  const saveJSON = () => {

    const json = JSON.stringify(data, null, 2);

    const blob = new Blob([json], { type: "application/json" });

    const url = URL.createObjectURL(blob);

    const link = document.createElement("a");

    link.href = url;

    link.download = "dane.json";

    link.click();

  };

  return (

    <div className="min-h-screen bg-gradient-to-br from-indigo-300 to-purple-400 p-10 flex justify-center items-start">

      <div className="w-full max-w-6xl bg-white/80 backdrop-blur-xl rounded-3xl shadow-2xl p-10 space-y-8 border border-white/40 text-center">

        <h1 className="text-4xl font-extrabold text-center text-gray-900 drop-shadow-sm">

          ⚙️ Dane osobowe

        </h1>

        {/* FORMULARZ I PRZYCISKI W DWÓCH KOLUMNACH */}

        <div className="grid grid-cols-1 md:grid-cols-2 gap-8">
```

Zapis, wyszukiwanie i odczyt – kompletny przykład

```
{/* FORMULARZ */}
          <div className="p-6 bg-white/60 backdrop-blur rounded-2xl shadow-lg border border-gray-200 space-y-4 flex flex-col items-center">
            <input
              type="text"
              name="name"
              placeholder="Imię"
              value={form.name}
              onChange={handleFormChange}
              className="w-full max-w-md p-3 rounded-xl border shadow-sm focus:ring-4 focus:ring-indigo-300 bg-white/70"
            />
            <input
              type="text"
              name="city"
              placeholder="Miasto"
              value={form.city}
              onChange={handleFormChange}
              className="w-full max-w-md p-3 rounded-xl border shadow-sm focus:ring-4 focus:ring-indigo-300 bg-white/70"
            />
            <input
              type="number"
              name="age"
              placeholder="Wiek"
              value={form.age}
              onChange={handleFormChange}
              className="w-full max-w-md p-3 rounded-xl border shadow-sm focus:ring-4 focus:ring-indigo-300 bg-white/70"
            />
            <input
              type="email"
              name="email"
              placeholder="Email"
              value={form.email}
              onChange={handleFormChange}
              className="w-full max-w-md p-3 rounded-xl border shadow-sm focus:ring-4 focus:ring-indigo-300 bg-white/70"
            />
            <input
              type="tel"
              name="phone"
              placeholder="Telefon"
              value={form.phone}
              onChange={handleFormChange}
              className="w-full max-w-md p-3 rounded-xl border shadow-sm focus:ring-4 focus:ring-indigo-300 bg-white/70"
            />
            <input
              type="text"
              name="occupation"
              placeholder="Zawód"
              value={form.occupation}
              onChange={handleFormChange}
              className="w-full max-w-md p-3 rounded-xl border shadow-sm focus:ring-4 focus:ring-indigo-300 bg-white/70"
            />
            <button
              onClick={handleAdd}
              className="w-full max-w-md bg-gradient-to-r from-blue-500 to-indigo-600 text-white rounded-xl p-3 shadow-lg hover:scale-105 transition font-bold"
            >
              ➕ Dodaj
            </button>
          </div>
          {/* PRZYCISKI */}
          <div className="p-6 bg-white/60 backdrop-blur rounded-2xl shadow-lg border border-gray-200 flex flex-col items-center justify-center space-y-4">
            <button
              onClick={saveJSON}
              className="bg-gradient-to-r from-green-500 to-emerald-600 text-white px-6 py-3 rounded-xl shadow-lg hover:scale-105 transition font-bold"
            >
              💾 Zapisz JSON
            </button>
          </div>
        </div>
```

Zapis, wyszukiwanie i odczyt – kompletny przykład

```
      {/* WYSZUKIWARKA */}
          <input
            placeholder="🔍 Wyszukaj wpis..."
            value={query}
            onChange={(e) => setQuery(e.target.value)}
            className="w-full max-w-md p-4 rounded-xl border shadow focus:ring-4
focus:ring-purple-300 bg-white/70 mx-auto"
          />
{/* TABELA */}
      <div className="overflow-x-auto rounded-2xl shadow-xl border border-gray-200 mx-
auto">
            <table className="w-full text-left border-collapse backdrop-blur-xl bg-white/80">
              <thead className="bg-indigo-300/60 text-gray-900 font-bold uppercase text-
sm">
                <tr>
                  <th className="p-4 border">ID</th>
                  <th className="p-4 border">Imię</th>
                  <th className="p-4 border">Miasto</th>
                  <th className="p-4 border">Wiek</th>
                  <th className="p-4 border">Email</th>
                  <th className="p-4 border">Telefon</th>
                  <th className="p-4 border">Zawód</th>
                  <th className="p-4 border">Akcje</th>
                </tr>
              </thead>
              <tbody>
                {filtered.length > 0 ? (
filtered.map((row) => (
  <tr
    key={row.id}
    className="hover:bg-gray-100/80 transition"
  >
    <td className="p-4 border text-center">{row.id}</td>
    {editingId === row.id ? (
      <>
        <td className="p-4 border">
          <input
            type="text"
            name="name"
            value={editForm.name}
            onChange={handleEditFormChange}
            className="w-full p-2 rounded border bg-white/70"
          />
        </td>
        <td className="p-4 border">
          <input
            type="text"
            name="city"
            value={editForm.city}
            onChange={handleEditFormChange}
            className="w-full p-2 rounded border bg-white/70"
          />
        </td>
        <td className="p-4 border">
          <input
            type="number"
            name="age"
            value={editForm.age}
            onChange={handleEditFormChange}
            className="w-full p-2 rounded border bg-white/70"
          />
        </td>
        <td className="p-4 border">
          <input
            type="email"
            name="email"
            value={editForm.email}
            onChange={handleEditFormChange}
            className="w-full p-2 rounded border bg-white/70"
          />
        </td>
        <td className="p-4 border">
          <input
            type="tel"
            name="phone"
            value={editForm.phone}
            onChange={handleEditFormChange}
            className="w-full p-2 rounded border bg-white/70"
          />
        </td>
```

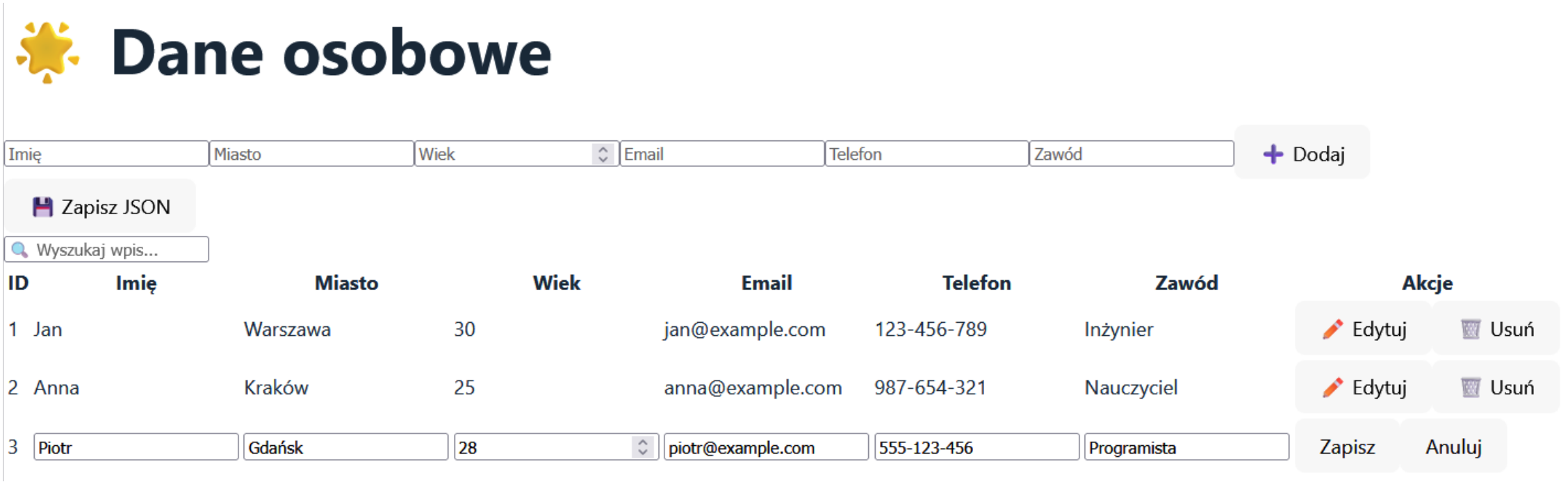
Zapis, wyszukiwanie i odczyt – kompletny przykład

```
<td className="p-4 border">
    <input
      type="text"
      name="occupation"
      value={editForm.occupation}
      onChange={handleEditFormChange}
      className="w-full p-2 rounded border bg-white/70"
    />
  </td>
<td className="p-4 border text-center space-x-2">
    <button
      onClick={handleSaveEdit}
      className="bg-green-500 text-white px-3 py-1 rounded hover:bg-green-600 transition"
    >
      Zapisz
    </button>
    <button
      onClick={handleCancelEdit}
      className="bg-gray-500 text-white px-3 py-1 rounded hover:bg-gray-600 transition"
    >
      Anuluj
    </button>
  </td>
</>
) : (
<>
  <td className="p-4 border">{row.name}</td>
  <td className="p-4 border">{row.city}</td>
  <td className="p-4 border text-center">{row.age}</td>
  <td className="p-4 border">{row.email}</td>
  <td className="p-4 border">{row.phone}</td>
  <td className="p-4 border">{row.occupation}</td>
  <td className="p-4 border text-center space-x-2">
    <button
      onClick={() => handleEdit(row)}
      className="bg-yellow-500 text-white px-3 py-1 rounded hover:bg-yellow-600 transition"
    >
      ✏️ Edytuj
    </button>
    <button
      onClick={() => handleDelete(row.id)}
      className="bg-red-500 text-white px-3 py-1 rounded hover:bg-red-600 transition"
    >
      🗑️ Usuń
    </button>
  </td>
</>
)}
</tr>
))
) : (
<tr>
  <td className="p-4 border text-center" colSpan="8">
    ❌ Brak wyników
  </td>
</tr>
)}
</tbody>
</table>
</div>
</div>
</div>
);

}
```

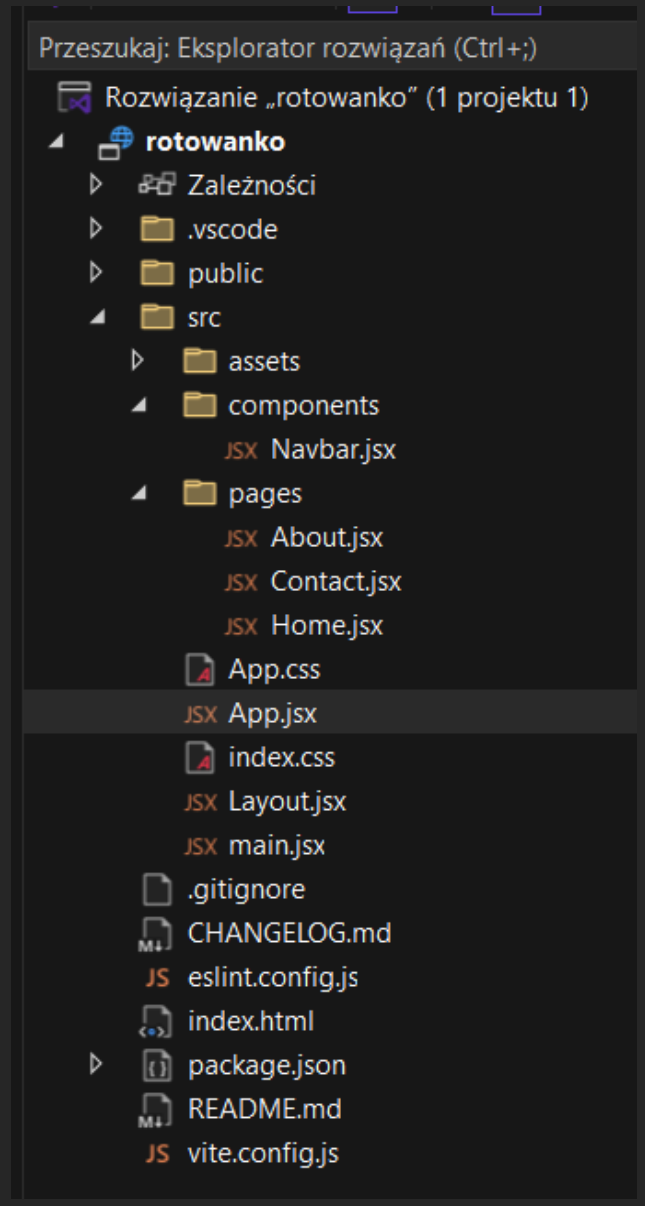
Opis

Opis działania

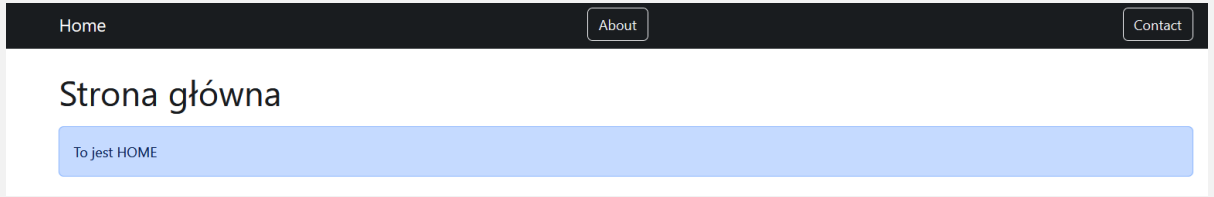
Aplikacja pozwala dodać dane do tabeli oraz do pliku. Przyciski w tabeli umożliwiają zmianę danych, zapisanie oraz usuwanie. Przykład można dostosować do własnych potrzeb.

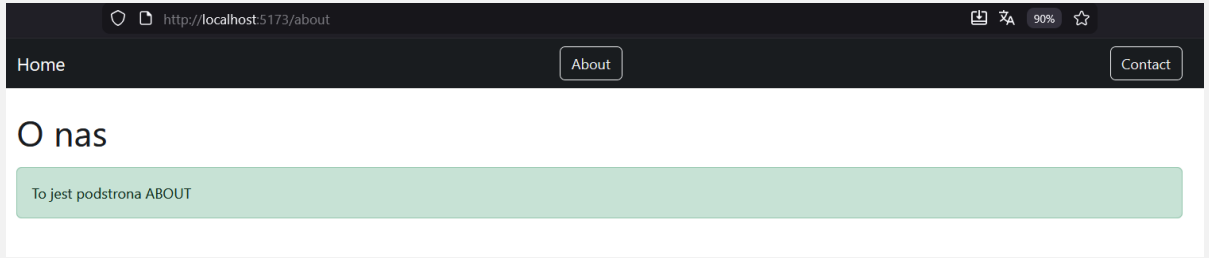


Routing z React



npm install react-router-dom - instalacja





Do czego użyć?
Do przełączania się pomiędzy podstronami w projekcie.

Routing w React

Contact.jsx | About.jsx | Home.jsx | App.jsx | main.jsx | Navbar.jsx | Co nowego?

Miscellaneous | { } "App" | default

```jsx
import { Routes, Route } from "react-router-dom"
import Layout from "./Layout"
import Home from "./pages/Home"
import About from "./pages/About"
import Contact from "./pages/Contact"

3 references
function App() {
    return (
        <Routes>
            <Route element={<Layout />}>
                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />
                <Route path="/contact" element={<Contact />} />
            </Route>
        </Routes>
    )
}

export default App
```

App.jsx

Routing w React

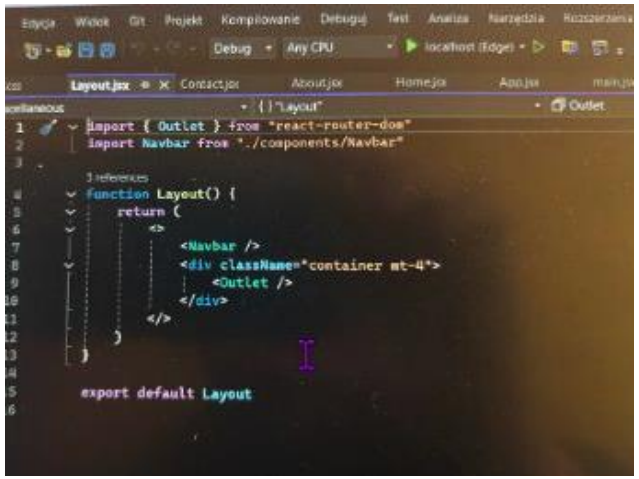
Contact.jsx | About.jsx | Home.jsx | App.jsx | main.jsx | **Navbar.jsx** | Co nowego?

Miscellaneous | { } "Navbar" | default

```jsx
import { Link } from "react-router-dom"

3 references
function Navbar() {
    return (
        <nav className="navbar navbar-dark bg-dark">
            <div className="container">
                <Link className="navbar-brand" to="/">Home</Link>
                <Link className="btn btn-outline-light me-2" to="/about">About</Link>
                <Link className="btn btn-outline-light" to="/contact">Contact</Link>
            </div>
        </nav>
    )
}

export default Navbar
```

Navbar z linkami

Layout z nawigacją



W projekcie użyto Bootstrap. Jak go zaimportować i użyć? Dowiesz się z przykładów na następnych stronach.

Routing w React

About.jsx | **Home.jsx** | App.jsx | main.jsx | Navbar.jsx | Co nowego?

{ } "Home" | default

```jsx
function Home() {
    return (
        <div>
            <h1>Strona główna</h1>
            <div className="alert alert-primary mt-3">
                To jest HOME
            </div>
        </div>
    )
}

export default Home
```

Przykładowa podstrona

Po jednym pliku na każdą podstronę.

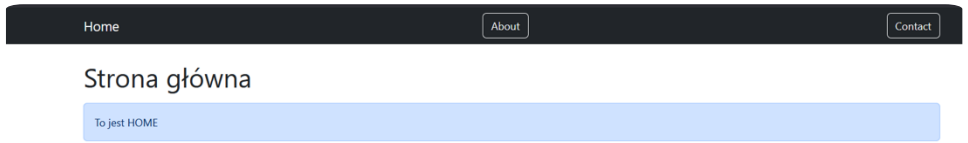
Contact.jsx | About.jsx | Home.jsx | App.jsx | **main.jsx** | Navbar.jsx | Co nowego?

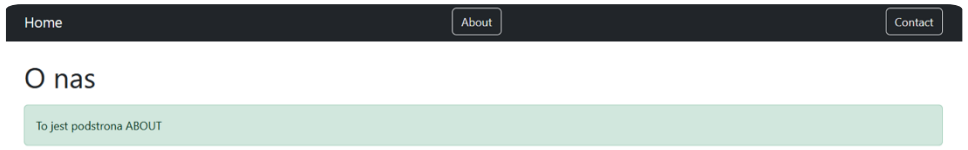
Miscellaneous | { } "main" | ReactDOM

```jsx
import React from "react"
import ReactDOM from "react-dom/client"
import { BrowserRouter } from "react-router-dom"
import App from "./App"
import "bootstrap/dist/css/bootstrap.min.css"

ReactDOM.createRoot(document.getElementById("root")).render(
    <BrowserRouter>
        <App />
    </BrowserRouter>
)
```

Main.jsx

Uruchamiamy główny komponent.





Efekt?

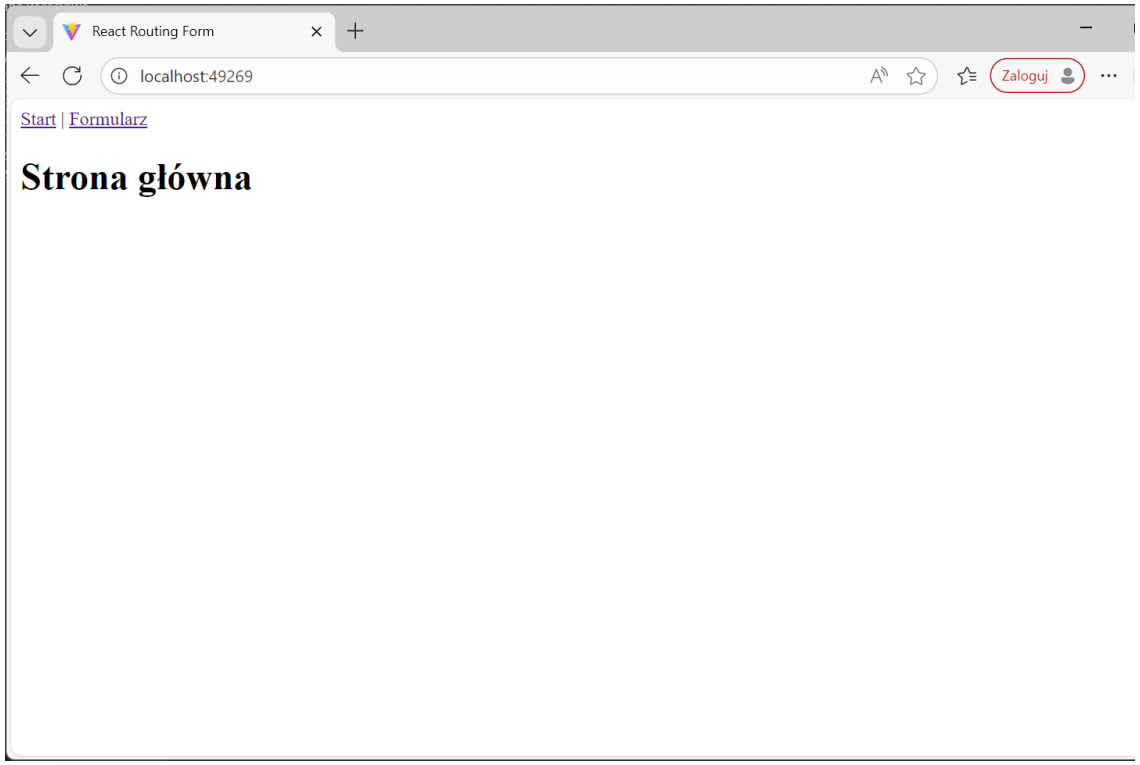
Prosty pasek, który prowadzi do konkretnej podstrony z zawartością.

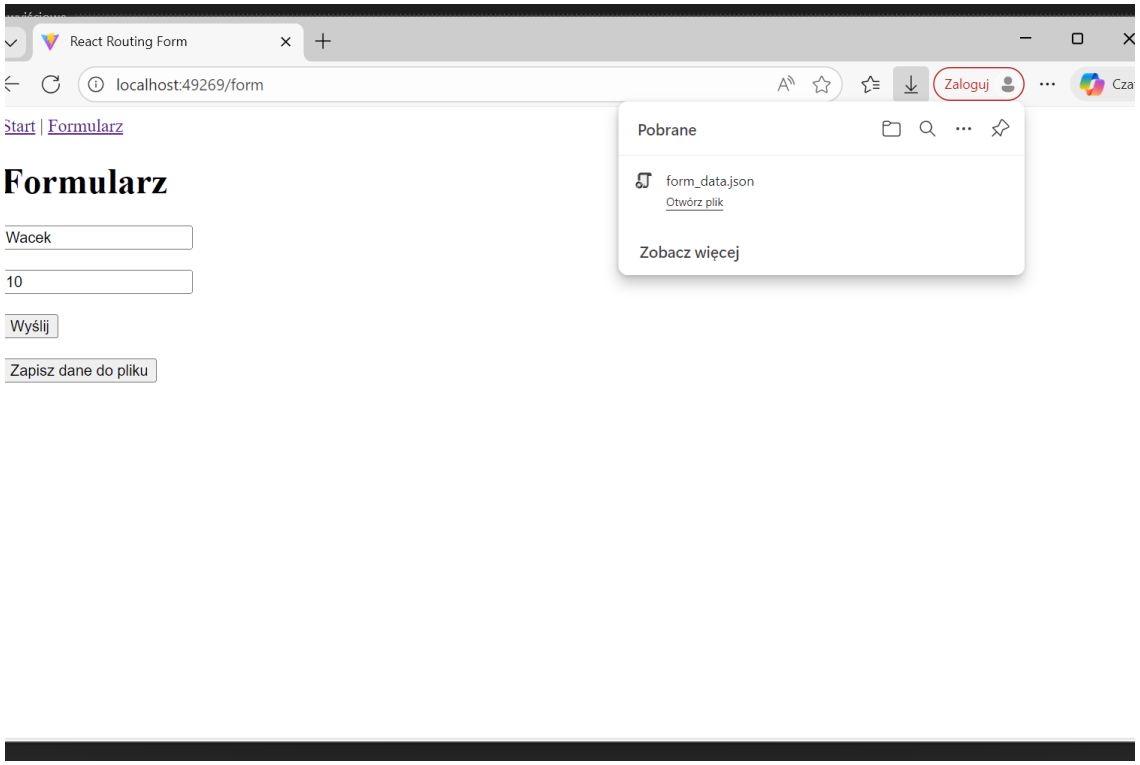
Zawartość wklejany na odpowiedniej podstronie.

Właściwości routingu:

Element	Rola
BrowserRouter	otacza aplikację
Routes	kontener tras
Route	pojedyncza trasa
Link	nawigacja
Outlet	miejsce na stronę

Zakładki, formularz, dane

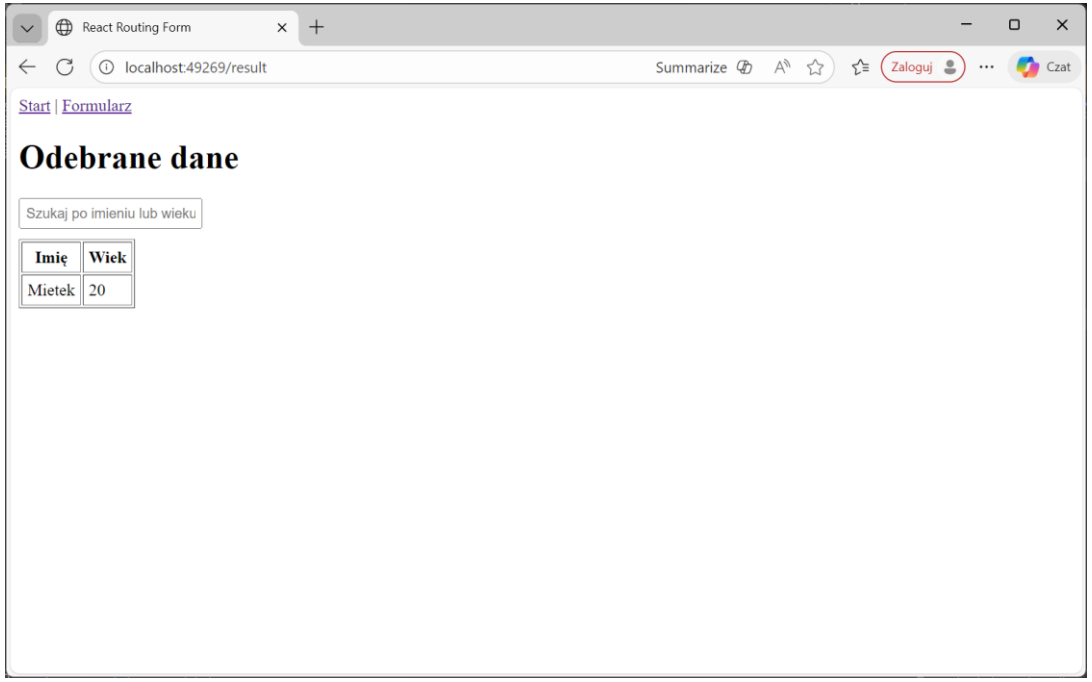




Zakładki, formularz, dane

Kompletna aplikacja, w której użytkownik wprowadza dane. Następnie zapisują się do pliku i mogą zostać wyświetlone przez wyszukiwanie.

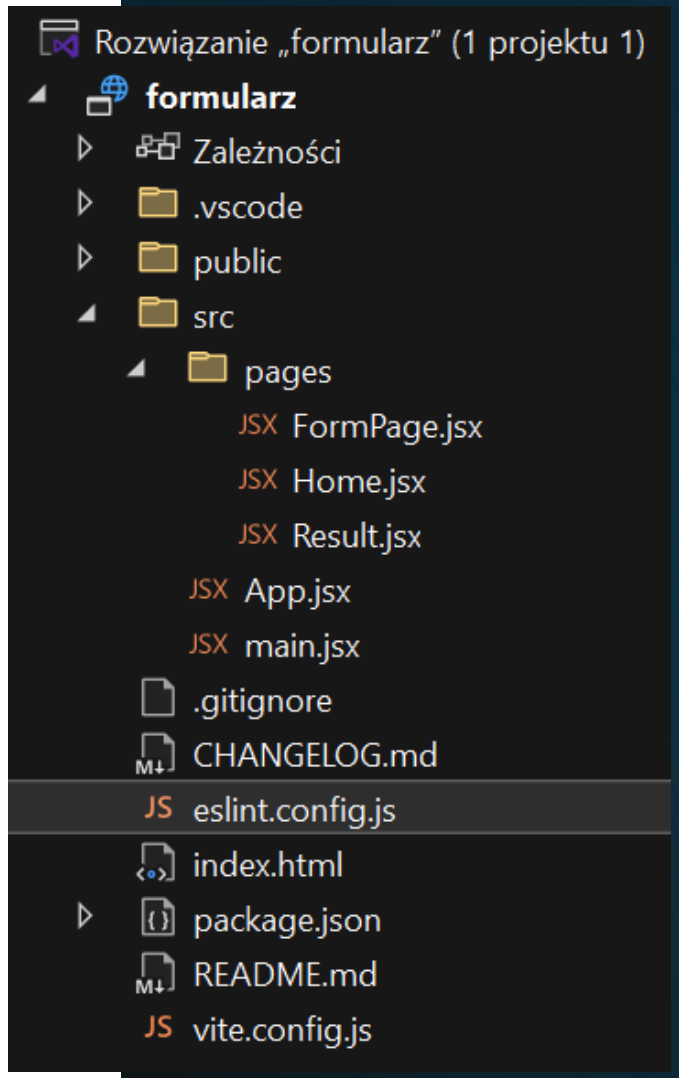
Paginacja wykorzystuje routing a dane wyświetlają się na kolejnej zakładce, do której nie ma bezpośredniego dostępu.



Zakładki, formularz, dane

Main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { BrowserRouter } from 'react-router-dom'
import App from './App'


ReactDOM.createRoot(document.getElementById('root')).rend
er(
<BrowserRouter>
<App />
</BrowserRouter>
)
```



App.jsx do projektu

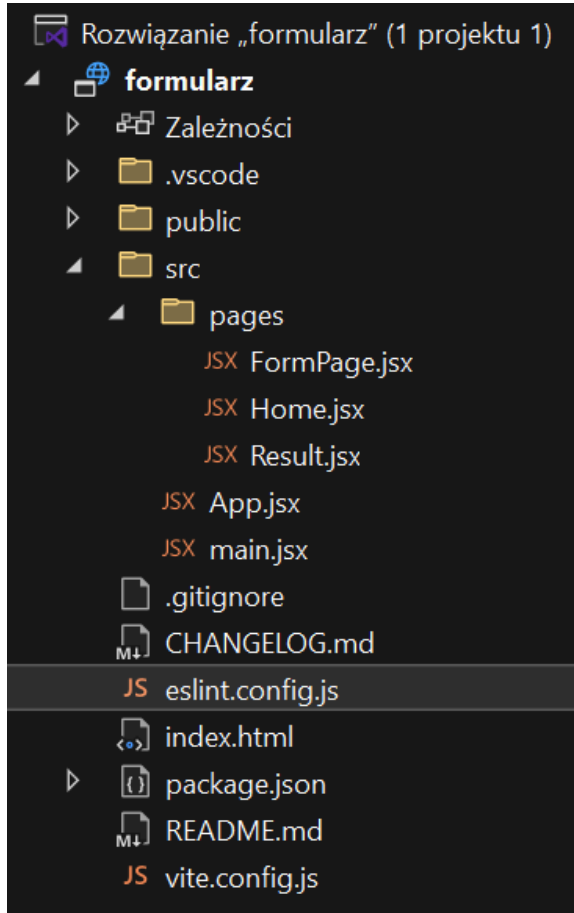
```
import { Routes, Route, Link } from 'react-router-dom'
import Home from './pages/Home'
import FormPage from './pages/FormPage'
import Result from './pages/Result'


export default function App() {
return (
<>
<nav>
<Link to="/">Start</Link> | <Link to="/form">Formularz</Link>
</nav>
<Routes>
<Route path="/" element={<Home />} />
<Route path="/form" element={<FormPage />} />
<Route path="/result" element={<Result />} />
</Routes>
</>
)
}
```

Pliki stron:

Home.jsx

```
export default function Home() {
return <h1>Strona główna</h1>
}
```

Oczywiście można dodać własne formatowanie. Projekt jedynie przedstawia sposób działania przykładowej aplikacji.



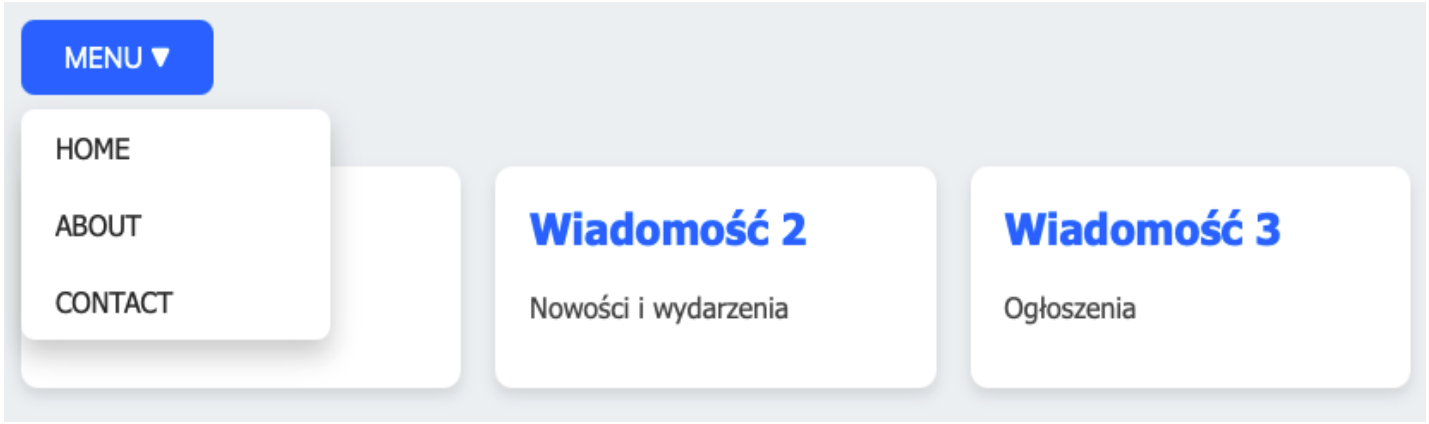
FormPage.jsx

```
import { useState } from "react"
import { useNavigate } from "react-router-dom"

function FormPage() {
const [name, setName] = useState("")
const [age, setAge] = useState("")
const [allData, setAllData] = useState([]) // nowa tablica
const navigate = useNavigate()


const handleSubmit = (e) => {
e.preventDefault()
const newEntry = { name, age }
const updatedData = [...allData, newEntry]
setAllData(updatedData)


// przekazujemy tablicę do Result
navigate("/result", { state: { data: updatedData } })
}

const handleSaveToFile = () => {
const data = [...allData, { name, age }]
const json = JSON.stringify(data, null, 2)
const blob = new Blob([json], { type:
"application/json" })
const url = URL.createObjectURL(blob)
const link = document.createElement("a")
link.href = url
link.download = "form_data.json"
document.body.appendChild(link)
link.click()
document.body.removeChild(link)
URL.revokeObjectURL(url)
}
return (
<div>
<h1>Formularz</h1>


<form onSubmit={handleSubmit}>
<input
type="text"
placeholder="Imię"
value={name}
onChange={(e) => setName(e.target.value)}
required
/>
<br /><br />
<input
type="number"
placeholder="Wiek"
value={age}
onChange={(e) => setAge(e.target.value)}
required
/>
<br /><br />
<button type="submit">Dodaj i pokaż</button>
</form>

<br />


<button onClick={handleSaveToFile}>Zapisz wszystkie dane do pliku</button>
</div>
)
}


export default FormPage
```

Result.jsx

```
import { useLocation } from "react-router-dom"
import { useState, useEffect } from "react"


function Result() {
const location = useLocation()
const data = location.state?.data || []


const [search, setSearch] = useState("")
const [filteredData, setFilteredData] = useState(data)


useEffect(() => {
setFilteredData(
data.filter(
(item) =>
item.name.toLowerCase().includes(search.toLowerCase()) ||
String(item.age).includes(search)
)
)
}, [search, data])

return (
<div>
<h1>Odebrane dane</h1>


{data.length === 0 ? (
<p>Brak danych (wejdź przez formularz)</p>
) : (
<>
<input
type="text"
placeholder="Szukaj po imieniu lub wieku"
value={search}
onChange={(e) => setSearch(e.target.value)}
style={{ marginBottom: "10px", padding: "5px" }}
/>


<table border="1" cellPadding="5">
<thead>
<tr>
<th>Imię</th>
<th>Wiek</th>
</tr>
</thead>
<tbody>
{filteredData.map((item, index) => (
<tr key={index}>
<td>{item.name}</td>
<td>{item.age}</td>
</tr>
))}
</tbody>
</table>
</>
)}
</div>
)
}


export default Result
```

Menu rozwijane



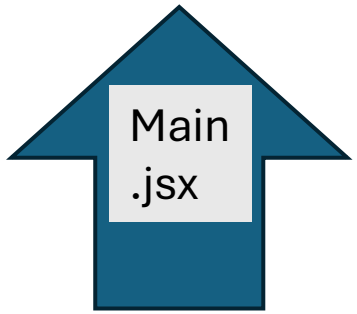
Projekt prostej aplikacji z Menu i kilkoma zakładkami w każdej podstronie.

App.jsx

```
import { useState } from "react";
import { Routes, Route, Link } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/Contact";

export default function App() {
  const [menuOpen, setMenuOpen] = useState(false);

  const toggleMenu = () => setMenuOpen(!menuOpen);

  return (
    <div className="App">
      <header>
        <div className="dropdown">
          <button className="dropbtn" onClick={toggleMenu}>
            MENU ▼
          </button>
          {menuOpen && (
            <div className="dropdown-content">
              <Link to="/" onClick={() => setMenuOpen(false)}>HOME</Link>
              <Link to="/about" onClick={() => setMenuOpen(false)}>ABOUT</Link>
              <Link to="/contact" onClick={() => setMenuOpen(false)}>CONTACT</Link>
            </div>
          )}
        </div>
      </header>

      <main>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </main>
    </div>
  );
}
```

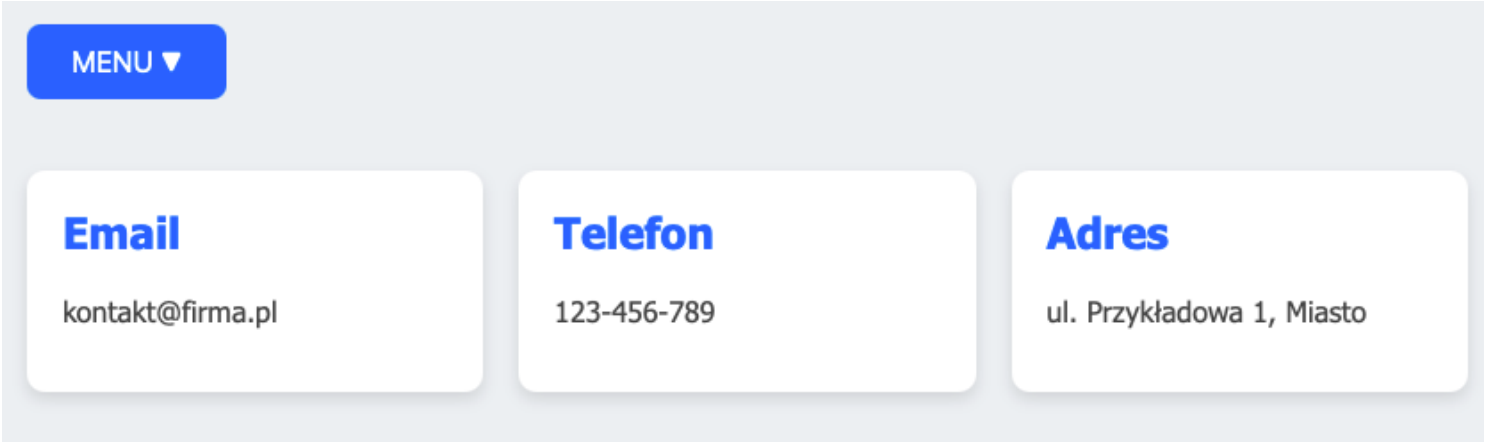
Menu rozwijane

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App";
import "./App.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```



Zawartość zakładek oczywiście może być dowolnie projektowana. Stosuje się układ kart w css i odpowiedni format kopiuje w kodzie.

Przykład css dla kart znajdziesz na następnej stronie.



CSS do projektu

```css
/* Global */
body {
  margin: 0;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: #f0f2f5;
}

.App {
  max-width: 900px;
  margin: 0 auto;
  padding: 20px;
}

/* Dropdown */
.dropdown {
  position: relative;
  display: inline-block;
  margin-bottom: 30px;
}

.dropbtn {
  background-color: #007bff;
  color: white;
  padding: 12px 25px;
  font-size: 16px;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background 0.3s;
}

.dropbtn:hover {
  background-color: #0056b3;
}

.dropdown-content {
  position: absolute;
  background-color: #ffffff;
  min-width: 180px;
  box-shadow: 0 8px 16px rgba(0,0,0,0.2);
  z-index: 1;
  border-radius: 8px;
  overflow: hidden;
  top: 50px;
}

.dropdown-content a {
  display: block;
  padding: 12px 20px;
  text-decoration: none;
  color: #333;
  transition: background 0.2s;
}

.dropdown-content a:hover {
  background-color: #f1f1f1;
}

/* Cards Layout for list */
.cards {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 20px;
  padding: 10px 0;
}

.card {
  background-color: #ffffff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0,0,0,0.1);
  transition: transform 0.2s, box-shadow 0.2s;
}

.card:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 16px rgba(0,0,0,0.2);
}

.card h2 {
  margin-top: 0;
  color: #007bff;
}

.card p {
  color: #555;
}
```

Przykładowa podstrona

```
export default function Contact() {
  const items = [
    { title: "Email", desc: "kontakt@firma.pl" },
    { title: "Telefon", desc: "123-456-789" },
    { title: "Adres", desc: "ul. Przykładowa 1, Miasto" }
  ];

  return (
    <div className="cards">
      {items.map((item, i) => (
        <div className="card" key={i}>
          <h2>{item.title}</h2>
          <p>{item.desc}</p>
        </div>
      ))}
    </div>
  );
}
```

Każda taka aplikacja wykorzystuje routing. Należy pamiętać o utrzymaniu odpowiedniej zależności plików i katalogów.

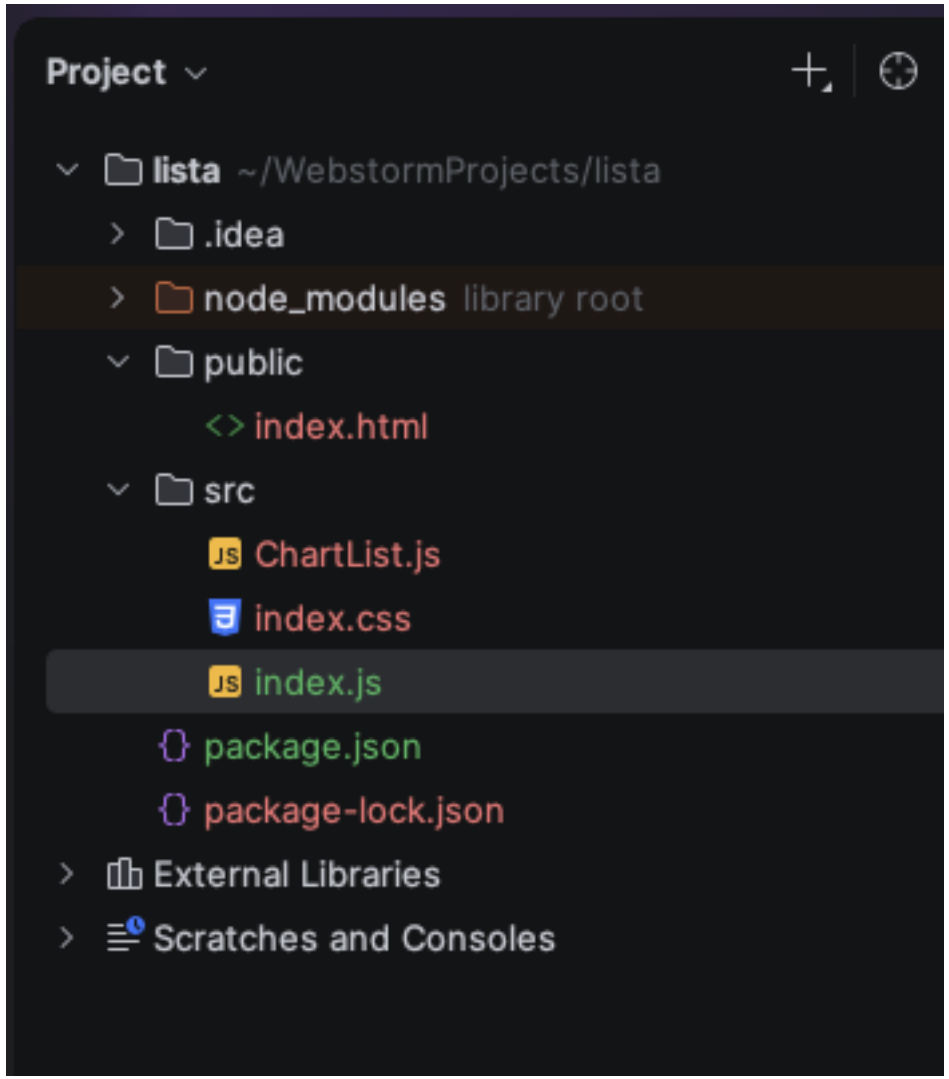
Prezentacja danych

React wykorzystuje wiele bibliotek do prezentacji danych na wykresach, tabelach i listach.

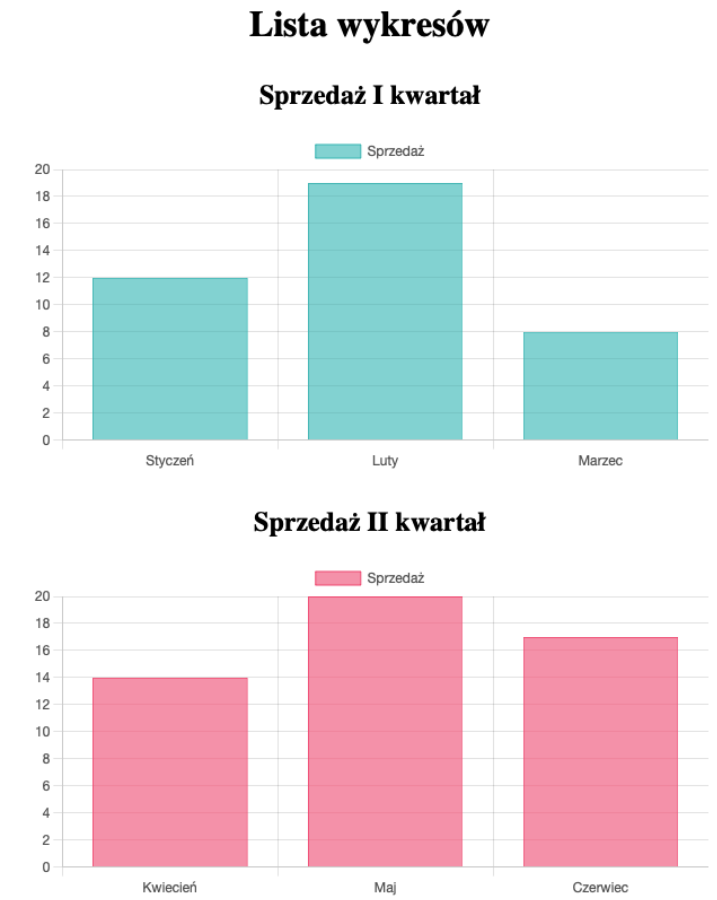
Już najprostsze z nich zapewniają wystarczające możliwości do przeniesienia danych na wykresy słupkowe czy osiowe.

	Możliwości w standardzie	Możliwości dostosowania	Próg wejścia	Trudność użycia
Chartist.js	małe	małe	niski	mała
Chart.js	przeciętne	małe	niski	mała
Recharts	przeciętne	przeciętne	niski	przeciętna
Google Charts	przeciętne	przeciętne	niski	przeciętna
Victory	przeciętne+	przeciętne	niski	przeciętna
ApexCharts	przeciętne+	przeciętne	niski	przeciętna
AmCharts	przeciętne+	przeciętne	niski	przeciętna
Highcharts	duże	duże	przeciętny	przeciętna
ECharts	bardzo duże	bardzo duże	przeciętny	duża

Wykresy i listy w React



Struktura i efekty



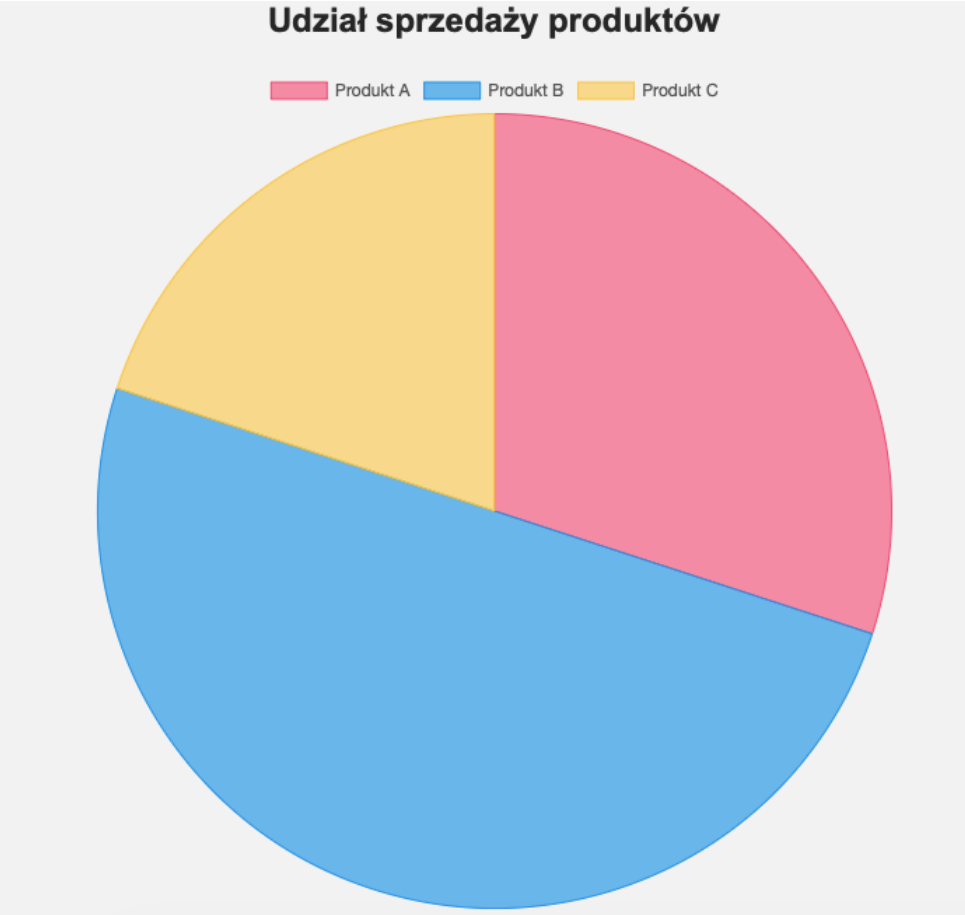
Wykresy i listy w React

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import ChartList from './ChartList';
import './index.css';

// Dane dla kilku wykresów
const charts = [
  {
    title: 'Sprzedaż I kwartał',
    data: {
      labels: ['Styczeń', 'Luty', 'Marzec'],
      datasets: [
        {
          label: 'Sprzedaż',
          data: [12, 19, 8],
          backgroundColor: 'rgba(75, 192, 192, 0.6)',
          borderColor: 'rgba(75, 192, 192, 1)',
          borderWidth: 1
        }
      ]
    },
    options: { responsive: true }
  },
  {
    title: 'Sprzedaż II kwartał',
    data: {
      labels: ['Kwiecień', 'Maj', 'Czerwiec'],
      datasets: [
        {
          label: 'Sprzedaż',
          data: [14, 20, 17],
          backgroundColor: 'rgba(255, 99, 132, 0.6)',
          borderColor: 'rgba(255, 99, 132, 1)',
          borderWidth: 1
        }
      ]
    },
    options: { responsive: true }
  }
];

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <h1 style={{ textAlign: 'center' }}>Lista wykresów</h1>
    <ChartList charts={charts} />
  </React.StrictMode>
);
```

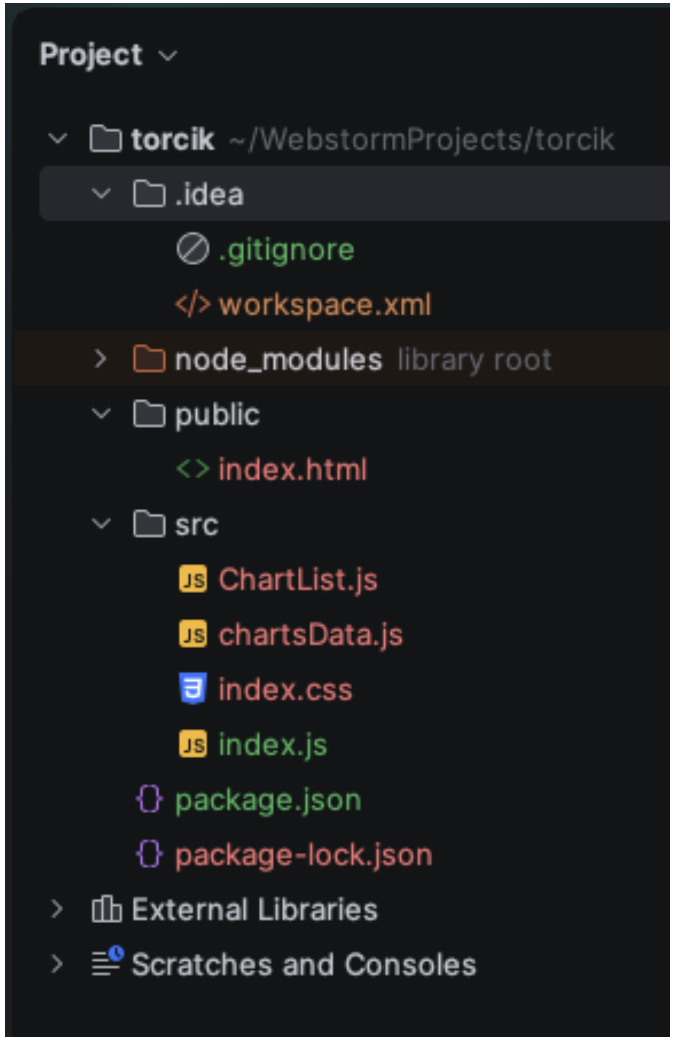
Pliki JavaScript

```
import React from 'react';
import { Bar } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  BarElement,
  Title,
  Tooltip,
  Legend
} from 'chart.js';

ChartJS.register(CategoryScale, LinearScale, BarElement, Title, Tooltip, Legend);

const ChartList = ({ charts }) => {
  return (
    <div>
      {charts.map((chart, idx) => (
        <div key={idx} style={{ width: '600px', margin: '30px auto', textAlign: 'center' }}>
          <h2>{chart.title}</h2>
          <Bar data={chart.data} options={chart.options} />
        </div>
      ))}
    </div>
  );
};

export default ChartList;
```

Wykres kołowy



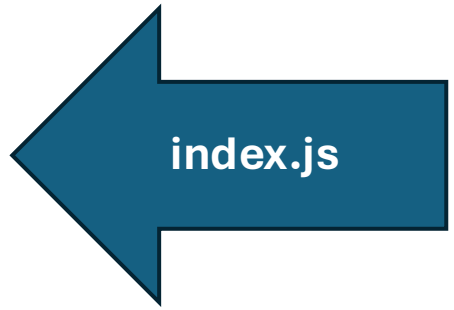
Efekt
i struktura



Pliki JavaScript do projektu

index.js | ChartList.js | chartsData.js | index.html | index.css | package.json

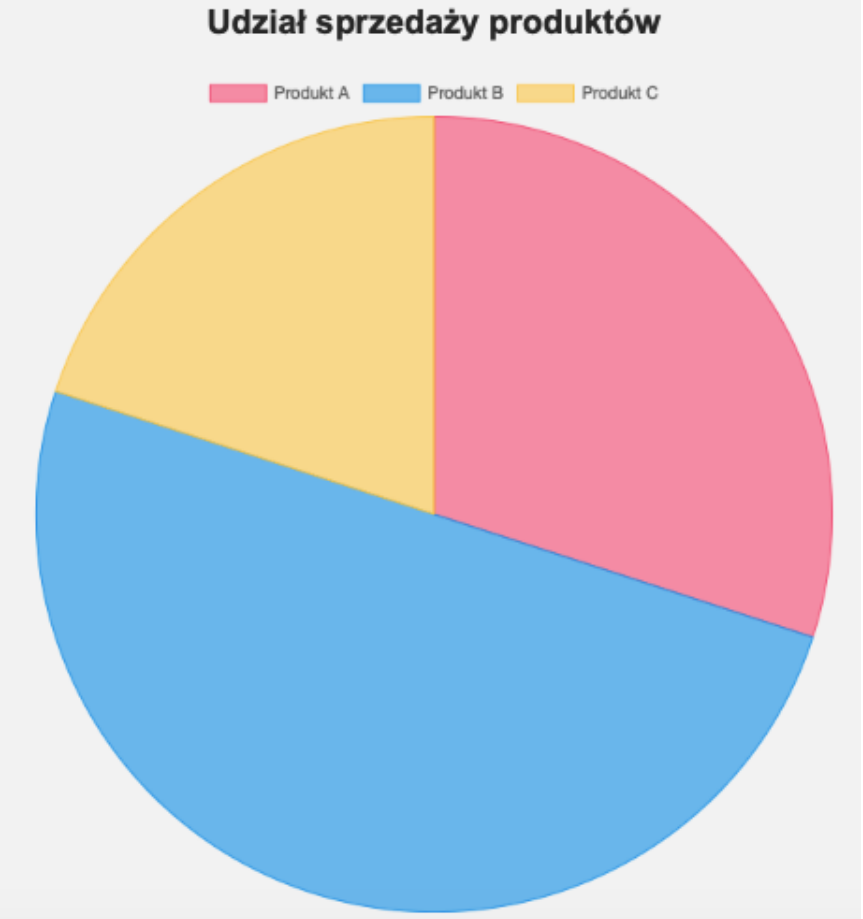
```
1   import React from 'react';
2   import ReactDOM from 'react-dom/client';
3   import ChartList from './ChartList';
4   import { charts } from './chartsData';
5   import './index.css';
6
7   const root : Root  = ReactDOM.createRoot(document.getElementById( elementId: 'root'));
8   root.render(
9       <React.StrictMode>
10          <h1 style={{ textAlign: 'center' }}>Lista wykresów</h1>
11          <ChartList charts={charts} />
12      </React.StrictMode>
13  );
```



ChartList i chartsData

```
import React from 'react';
import { Bar, Pie } from 'react-chartjs-2';
import { Chart as ChartJS, CategoryScale, LinearScale, BarElement, ArcElement, Title, Tooltip, Legend } from 'chart.js';

// Rejestracja Chart.js
ChartJS.register(CategoryScale, LinearScale, BarElement, ArcElement, Title, Tooltip, Legend);

const ChartList = ({ charts }) => {
  return (
    <div>
      {charts.map((chart, idx) => (
        <div key={idx} style={{ width: '600px', margin: '30px auto', textAlign: 'center' }}>
          <h2>{chart.title}</h2>
          {chart.type === 'bar' && <Bar data={chart.data} options={chart.options} />}
          {chart.type === 'pie' && <Pie data={chart.data} options={chart.options} />}
        </div>
      ))}
    </div>
  );
};

export default ChartList;
```

```
export const charts = [
  {
    title: 'Sprzedaż I kwartał',
    type: 'bar',
    data: {
      labels: ['Styczeń', 'Luty', 'Marzec'],
      datasets: [
        {
          label: 'Sprzedaż',
          data: [12, 19, 8],
          backgroundColor: 'rgba(75, 192, 192, 0.6)',
          borderColor: 'rgba(75, 192, 192, 1)',
          borderWidth: 1
        }
      ]
    },
    options: { responsive: true }
  },
  {
    title: 'Sprzedaż II kwartał',
    type: 'bar',
    data: {
      labels: ['Kwiecień', 'Maj', 'Czerwiec'],
      datasets: [
        {
          label: 'Sprzedaż',
          data: [14, 20, 17],
          backgroundColor: 'rgba(255, 99, 132, 0.6)',
          borderColor: 'rgba(255, 99, 132, 1)',
          borderWidth: 1
        }
      ]
    },
    options: { responsive: true }
  },
  {
    title: 'Udział sprzedaży produktów',
    type: 'pie',
    data: {
      labels: ['Produkt A', 'Produkt B', 'Produkt C'],
      datasets: [
        {
          label: 'Sprzedaż',
          data: [30, 50, 20],
          backgroundColor: [
            'rgba(255, 99, 132, 0.6)',
            'rgba(54, 162, 235, 0.6)',
            'rgba(255, 206, 86, 0.6)'
          ],
          borderColor: [
            'rgba(255, 99, 132, 1)',
            'rgba(54, 162, 235, 1)',
            'rgba(255, 206, 86, 1)'
          ],
          borderWidth: 1
        }
      ]
    },
    options: { responsive: true }
  }
];
```

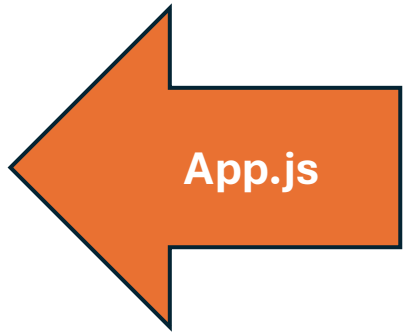


Efekt?

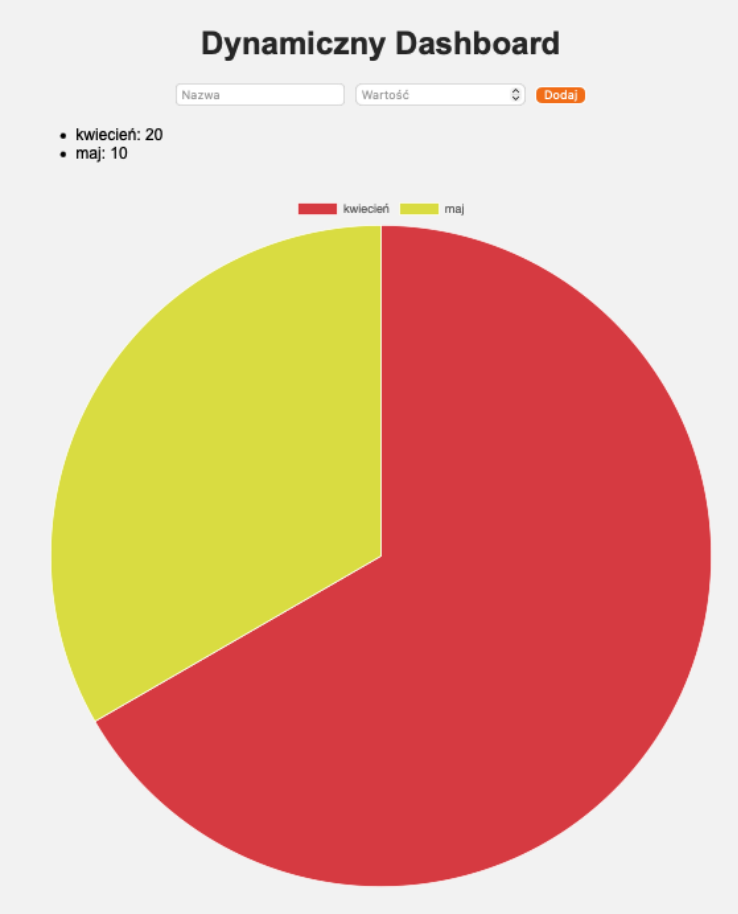
Przedstawiony kod to kontynuacja poprzedniego przykładu z wykresem słupkowym. Aplikacja przedstawiająca dane zwykle zawiera listę lub formularz do wprowadzania danych.

Dynamiczny formularz, lista i wykres kołowy

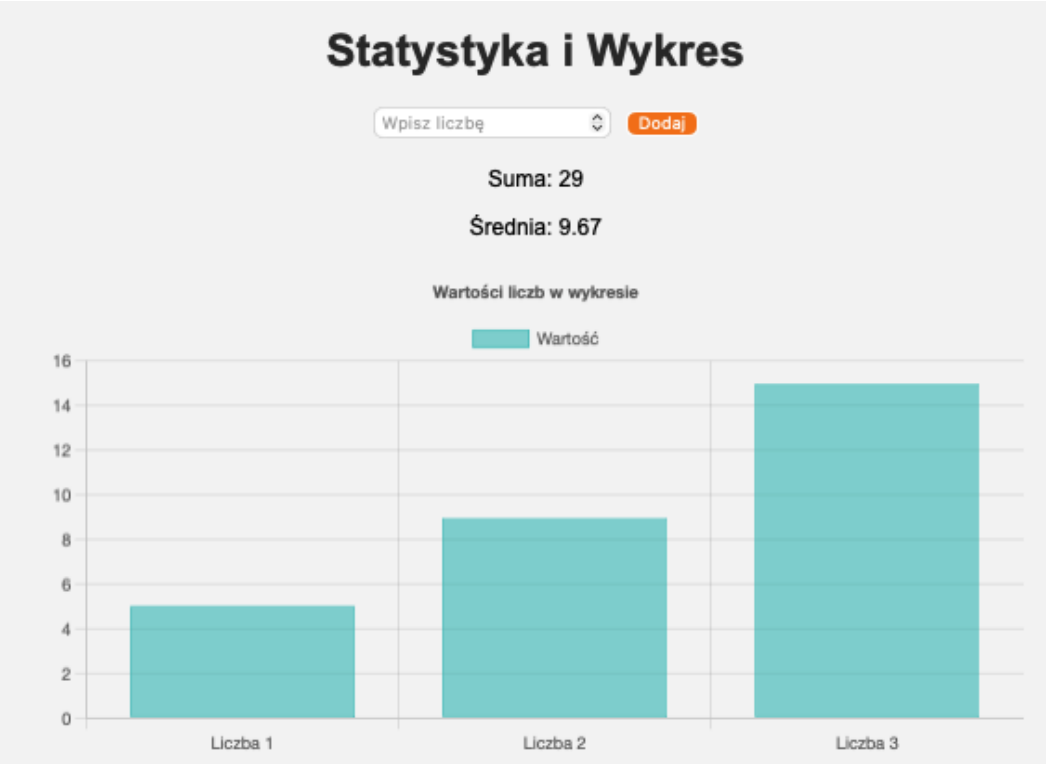
```
import React, { useState } from 'react';
import { Pie } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  ArcElement,
  Tooltip,
  Legend
} from 'chart.js';

// Rejestracja Chart.js
ChartJS.register(ArcElement, Tooltip, Legend);

const App = () => {
  const [items, setItems] = useState([]);
  const [name, setName] = useState('');
  const [value, setValue] = useState('');

  // Dodawanie nowego wpisu
  const handleAdd = (e) => {
    e.preventDefault();
    if (name && value) {
      setItems([...items, { name, value: Number(value) }]);
      setName('');
      setValue('');
    }
  };

  // Przygotowanie danych do wykresu Pie
  const pieData = {
    labels: items.map(item => item.name),
    datasets: [
      {
        label: 'Wartości',
        data: items.map(item => item.value),
        backgroundColor: items.map(
          (_, idx) => `hsl(${(idx * 60) % 360}, 70%, 60%)`
        ),
        borderColor: 'white',
        borderWidth: 1
      }
    ]
  };

  const pieOptions = {
    responsive: true,
    plugins: {
      legend: { position: 'top' },
      title: { display: true, text: 'Dynamiczny wykres Pie' }
    }
  };

  return (
    <div style={{ maxWidth: '700px', margin: '20px auto', textAlign: 'center' }}>
      <h1>Dynamiczny Dashboard</h1>

      {/* Formularz */}
      <form onSubmit={handleAdd} style={{ marginBottom: '20px' }}>
        <input
          type="text"
          placeholder="Nazwa"
          value={name}
          onChange={e => setName(e.target.value)}
          style={{ marginRight: '10px' }}
        />
        <input
          type="number"
          placeholder="Wartość"
          value={value}
          onChange={e => setValue(e.target.value)}
          style={{ marginRight: '10px' }}
        />
        <button type="submit">Dodaj</button>
      </form>

      {/* Lista danych */}
      <ul style={{ textAlign: 'left', marginBottom: '30px' }}>
        {items.map((item, idx) => (
          <li key={idx}>{item.name}: {item.value}</li>
        ))}
      </ul>

      {/* Wykres Pie */}
      {items.length > 0 && <Pie data={pieData} options={pieOptions} />}
    </div>
  );
};

export default App;
```



Dynamiczny formularz, lista i wykres kołowy



To fragment typowej aplikacji pozwalającej wpisać i gromadzić dane. W wersji rozszerzonej stosuje się zestawienia w postaci tabeli, wykresy liniowe oraz funkcje statystyczne.



Prosta funkcja statystyka i dynamiczny wykres

```
import React, { useState } from 'react';
import { Bar } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  BarElement,
  Title,
  Tooltip,
  Legend
} from 'chart.js';

ChartJS.register(CategoryScale, LinearScale, BarElement, Title, Tooltip, Legend);

const App = () => {
  const [numbers, setNumbers] = useState([]);
  const [input, setInput] = useState('');

  // Dodanie liczby do listy
  const handleAdd = (e) => {
    e.preventDefault();
    const num = Number(input);
    if (!isNaN(num)) {
      setNumbers([...numbers, num]);
      setInput('');
    }
  };

  // Funkcje statystyczne
  const sum = numbers.reduce((acc, n) => acc + n, 0);
  const average = numbers.length > 0 ? sum / numbers.length : 0;

  // Dane do wykresu
  const chartData = {
    labels: numbers.map((_, idx) => ` Liczba ${idx + 1}`),
    datasets: [
      {
        label: 'Wartość',
        data: numbers,
        backgroundColor: 'rgba(75,192,192,0.6)',
        borderColor: 'rgba(75,192,192,1)',
        borderWidth: 1
      }
    ]
  };

  const chartOptions = {
    responsive: true,
    plugins: {
      legend: { position: 'top' },
      title: { display: true, text: 'Wartości liczb w wykresie' }
    }
  };

  return (
    <div style={{ maxWidth: '700px', margin: '20px auto', textAlign: 'center' }}>
      <h1>Statystyka i Wykres</h1>

      {/* Formularz */}
      <form onSubmit={handleAdd} style={{ marginBottom: '20px' }}>
        <input
          type="number"
          placeholder="Wpisz liczbę"
          value={input}
          onChange={(e) => setInput(e.target.value)}
          style={{ marginRight: '10px' }}
        />
        <button type="submit">Dodaj</button>
      </form>

      {/* Statystyka */}
      <div style={{ marginBottom: '20px' }}>
        <p>Suma: {sum}</p>
        <p>Średnia: {average.toFixed(2)}</p>
      </div>

      {/* Wykres */}
      {numbers.length > 0 && <Bar data={chartData} options={chartOptions} />}
    </div>
  );
};

export default App;
```

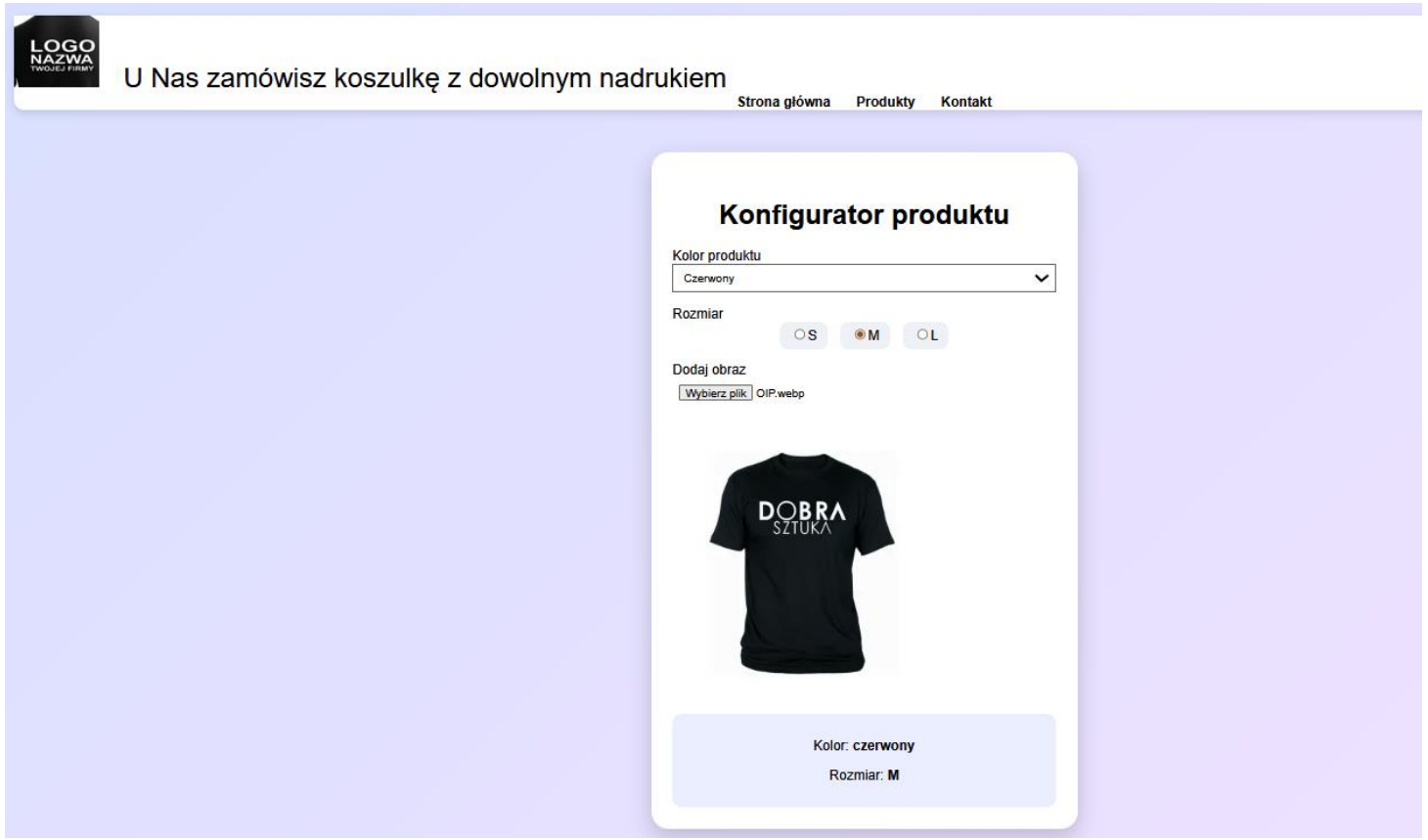
App.js

Zadania do samodzielnego wykonania

Zadanie 1

Do wykonanego w dziale pierwszym zadania dodaj estetyczny nagłówek według wzoru oraz wyśrodkowane menu.

... ale to jeszcze nie koniec ...



Zadania do samodzielnego wykonania

Rozwiń projekt stosując dowolne formatowanie a następnie:

- na karcie „produkty” dodaj cennik w postaci listy: S-50 zł, M-55 zł, L-60 zł, nadruk -20 zł
- na kacie „kontakt” dodaj telefon: 888-000-123, e-mail: koszulkowo@wp.pl.

Użytkownik będzie przełączał się pomiędzy kartami.
W projekcie wykorzystaj routing. Możesz wykorzystać dowolny styl css – napisany przez ciebie lub gotowe klasy z zaimportowanej biblioteki.

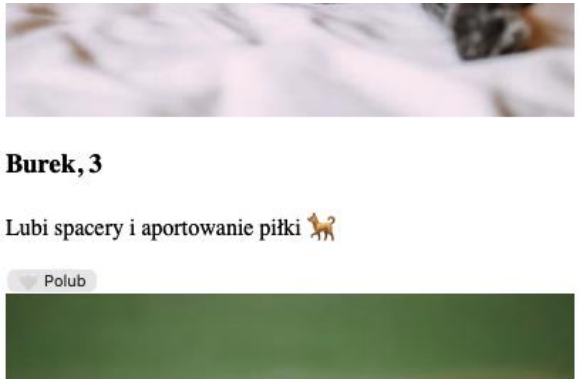
Zadania do samodzielnego wykonania

Zadanie 2

Przed Tobą zadanie w formie aplikacji „Psinder”. Składa się z trzech zakładek (Psy-lista psów, Polubienia – lista polubieni, O aplikacji – opis programu).

Jak to działa? Użytkownik daje polubienia psom. Lista polubień znajduje się na drugiej karcie.

Zadania do samodzielnego wykonania



Po naciśnięciu przycisku


```
<div className="bg-white rounded-2xl shadow-lg overflow-hidden">
 <img src={dog.img} alt={dog.name} className="w-full h-48 object-cover" />
 <div className="p-4">
  <h3 className="text-lg font-bold">{dog.name}, {dog.age}</h3>
  <p className="text-gray-600 text-sm mt-1">{dog.desc}</p>
  <button
   onClick={() => toggleLike(dog)}
   className={` btn-like ${liked.find(d => d.id === dog.id) ? 'btn-like-liked' : 'btn-like-
unliked'}` }
  >
   {liked.find(d => d.id === dog.id) ? "❤️ Polubiono" : "♡ Polub"}
  </button>
 </div>
</div>
```

Jedna karta

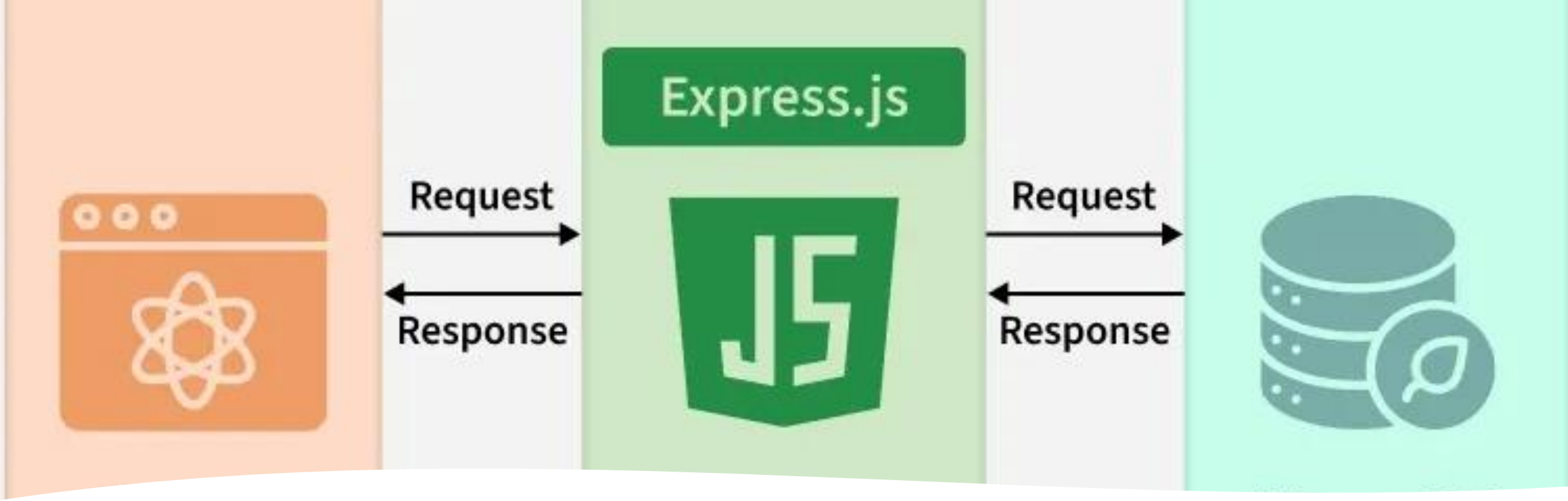
Express – framework jako backend

Framework wykorzystywany jest do tworzenia prostych stron internetowych

oraz rozbudowanych hybrydowych aplikacji webowych, z możliwością uruchomienia w przeglądarce.

Usprawnia pracę i rozszerza funkcjonalności Node.js.

Express.JS umożliwia również tworzenie czatów na żywo, ze względu na obsługę żądań w czasie rzeczywistym. Nadaje się również do zarządzania ciasteczkami, logowania do serwisu oraz pracy w trybie sesji.



Express

Express działa po stronie **backendu (serwera)**, a **React** po stronie **frontendu (interfejsu użytkownika)**.

Najczęściej Express wykorzystuje się do **tworzenia API dla aplikacji React.** React wysyła zapytania do serwera, a Express je obsługuje.

Express

Łączenie React z bazą danych.

Express może komunikować się z bazą danych, np.:
- **MongoDB**
- **PostgreSQL**
- **MySQL**

React nie powinien łączyć się z bazą bezpośrednio – robi to backend (np. Express)

Przykład:

- React: wysyła zapytanie o listę użytkowników
- Express: pobiera dane z bazy i odsyła JSON.

Express app.get("/api/users", (req, res) => { res.json([{ name: "Jan" }, { name: "Anna" }]); });

React potem pobiera dane np. przez fetch lub axios.


```
mkdir express-test
cd express-test
npm init -y
npm install express
```

API i React Vite - lokalnie

React (także React + Vite)
NIE tworzy API – React jest **frontendem**.
API robi się w **backendzie** (np. Node + Express) a React **korzysta z API** przez fetch / axios.

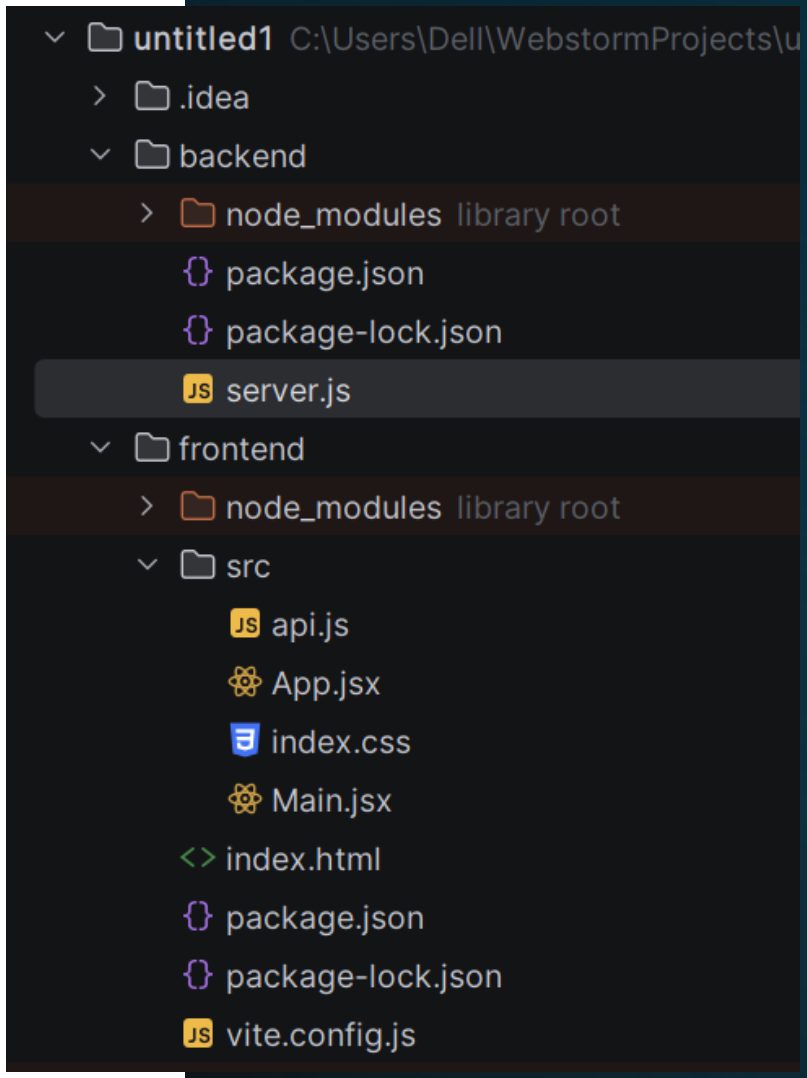


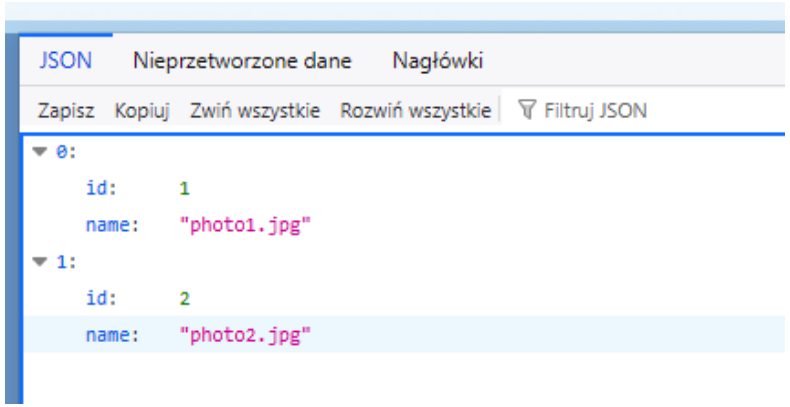
Photo API App

Photo name Add

- photo1.jpg Delete
- photo2.jpg Delete
- Photo1 Delete

Server.js

```
import express from "express";
import cors from "cors";

const app = express();
const PORT = 3000;
app.use(cors());
app.use(express.json());
let photos = [
  { id: 1, name: "photo1.jpg" },
  { id: 2, name: "photo2.jpg" }
];
// GET wszystkie zdjęcia
app.get("/photos", (req, res) => {
  res.json(photos);
});
// POST nowe zdjęcie
app.post("/photos", (req, res) => {

  const newPhoto = {
    id: Date.now(),
    name: req.body.name
  };
  photos.push(newPhoto);
  res.status(201).json(newPhoto);
});
// DELETE zdjęcie
app.delete("/photos/:id", (req, res) => {
  const id = parseInt(req.params.id);
  photos = photos.filter(p => p.id !== id);
  res.json({ message: "Deleted" });
});
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```



App.jsx

```
import { useEffect, useState } from "react";

import { getPhotos, addPhoto, deletePhoto } from "./api";


function App() {

const [photos, setPhotos] = useState([]);

const [name, setName] = useState("");


const loadPhotos = async () => {

const data = await getPhotos();

setPhotos(data);

};


useEffect(() => {

loadPhotos();

}, []);


const handleAdd = async () => {

if (!name) return;

await addPhoto(name);

setName("");

loadPhotos();

};


const handleDelete = async (id) => {

await deletePhoto(id);

loadPhotos();

};


return (

<div style={{ padding: 20 }}>

<h1>Photo API App</h1>


<input

value={name}

onChange={(e) => setName(e.target.value)}

placeholder="Photo name"

/>

<button onClick={handleAdd}>Add</button>


<ul>

{photos.map((photo) => (

<li key={photo.id}>

{photo.name}

<button onClick={() => handleDelete(photo.id)}>

Delete

</button>

</li>

))}

</ul>

</div>

);

}


export default App;
```

API.js

```
const API_URL = "http://localhost:3000/photos";

export async function getPhotos() {
  const res = await fetch(API_URL);
  return res.json();
}

export async function addPhoto(name) {
  const res = await fetch(API_URL, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ name })
  });
  return res.json();
}

export async function deletePhoto(id) {
  await fetch(`${API_URL}/${id}`, {
    method: "DELETE"
  });
}
```

Co to jest?

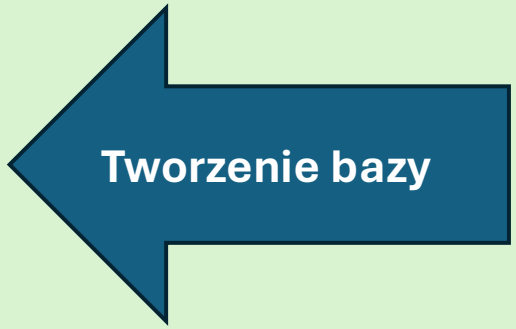
Prosta aplikacja w formie listy, do której użytkownik może dopisywać kolejne dane. Oczywiście projekt można rozwijać dodając wyszukiwarkę, dodatkowe karty lub zdjęcia.

Przechowywanie danych to nie tylko JSON.

SQLite i React

Dodaj do folderu backend plik database.js. Następnie wpisz w niego:

```
import sqlite3 from "sqlite3";
const db = new sqlite3.Database("./photos.db", (err) => {
  if (err) {
    console.error(err.message);
  } else {
    console.log("Connected to SQLite database.");
  }
});
export default db;
```



Tworzenie tabeli i połączenie z bazą

Dodaj do database.js:

```
db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS photos (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT
    )
  `);
});
```

Dodaj do pliku server.js:

```
import db from "./database.js";
```

CRUD na bazie danych

W pliku server.js kolejno wklejamy kolejne frameny zależne od rodzaju zapytania.

```
app.get("/photos", (req, res) => {
  db.all("SELECT * FROM photos", [], (err, rows) => {
    if (err) {
      res.status(500).json(err);
      return;
    }
    res.json(rows);
  });
});
```

```
app.post("/photos", (req, res) => {
  const { name } = req.body;

  db.run(
    "INSERT INTO photos(name) VALUES(?)",
    [name],
    function (err) {
      if (err) {
        res.status(500).json(err);
        return;
      }

      res.json({
        id: this.lastID,
        name
      });
    }
  );
});
```

CRUD na bazie danych

```
app.delete("/photos/:id", (req, res) => {
  const id = req.params.id;

  db.run("DELETE FROM photos WHERE id=?", id, (err) => {
    if (err) {
      res.status(500).json(err);
      return;
    }

    res.json({ message: "Deleted" });
  });
});
```

Ważne!
Pamiętaj aby włączyć backend. Tylko wtedy cały proces zakończy się powodzeniem.

Dodatkowo, do podglądu bazy danych może użyć narzędzi wspomagających zarządzanie bazą, jak np. DB Browser for SQLite.

DB Browser for SQLite

About **Download** Blog Docs GitHub Gitter Patreon

Downloads

(**Please** consider sponsoring us on Patreon 😄)

Windows

Our latest release (3.13.1) for Windows:

- DB Browser for SQLite - Standard installer for 32-bit Windows
- DB Browser for SQLite - .zip (no installer) for 32-bit Windows
- DB Browser for SQLite - Standard installer for 64-bit Windows
- DB Browser for SQLite - .zip (no installer) for 64-bit Windows

Free code signing provided by SignPath.io, certificate by SignPath Foundation.

Multer

npm install multer

backend/uploads

Multer to popularne oprogramowanie pośredniczące (middleware) dla Node.js i Express.js, stworzone do obsługi przesyłania plików (multipart/form-data). Charakteryzuje się wysoką wydajnością, opartą na busboy (specjalny silnik), i pozwala na zapisywanie plików bezpośrednio na dysku lub w pamięci.

Plik server.js do przesyłu plików

```
import express from "express";
import cors from "cors";
import multer from "multer";

const app = express();
const PORT = 3000;

app.use(cors());
app.use(express.json());

// udostępnienie folderu ze zdjęciami
app.use("/uploads", express.static("uploads"));

/* konfiguracja uploadu */
const storage = multer.diskStorage({
 destination: (req, file, cb) => {
  cb(null, "uploads/");
 },
 filename: (req, file, cb) => {
  cb(null, Date.now() + "-" + file.originalname);
 }
});

const upload = multer({ storage });

/* baza w pamięci */
let photos = [];

/* endpoint uploadu */
app.post("/upload", upload.single("photo"), (req, res) => {

 const photo = {
  id: Date.now(),
  filename: req.file.filename
 };

 photos.push(photo);

 res.json(photo);
});

/* endpoint listy zdjęć */
app.get("/photos", (req, res) => {
 res.json(photos);
});

/* endpoint pojedynczego zdjęcia */
app.get("/photo/:name", (req, res) => {
 res.sendFile(req.params.name, { root: "uploads" });
});

app.listen(PORT, () => {
 console.log(` Server running on http://localhost:${PORT}` );
});
```

Endpointy

POST http://localhost:3000/upload

http://localhost:3000/uploads/171000000-cat.jpg

GET http://localhost:3000/photos


```
[
 {
  "id": 171000000,
  "filename": "171000000-cat.jpg"
 }
]
```

Wyświetlenie zdjęcia – API.js

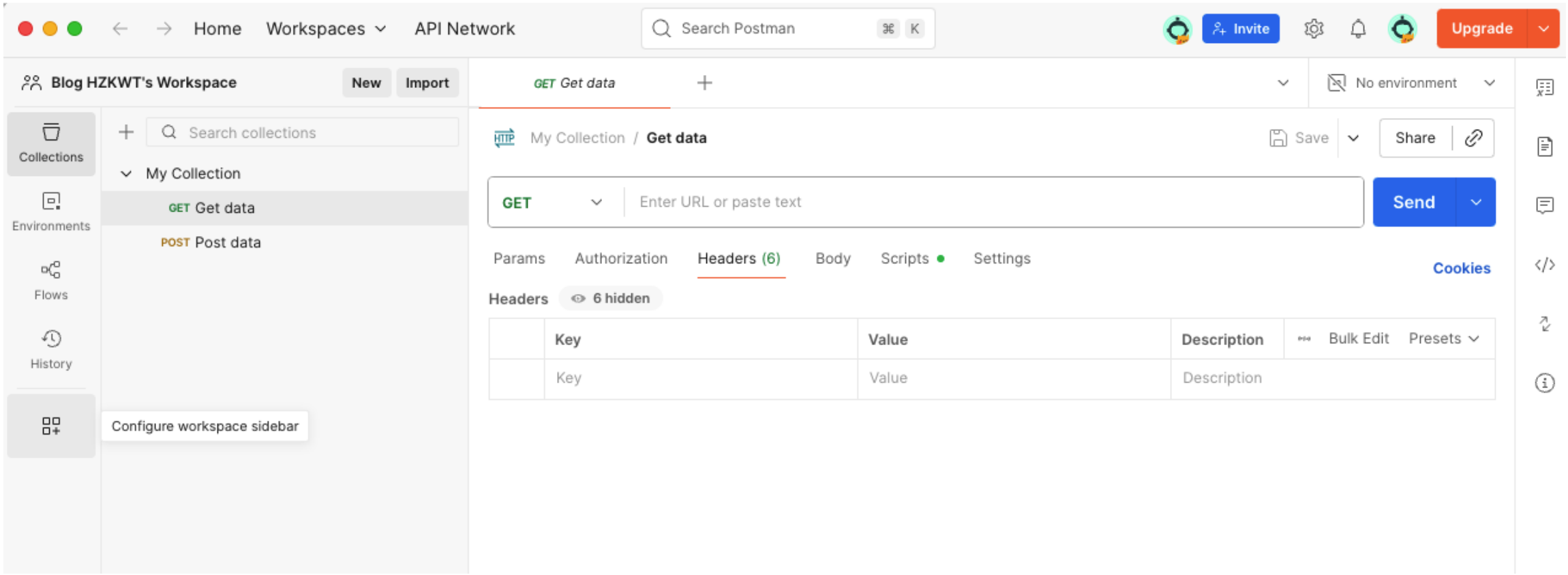
```
export async function uploadPhoto(file) {

  const formData = new FormData();
  formData.append("photo", file);

  const res = await
fetch("http://localhost:3000/upload", {
    method: "POST",
    body: formData
  });

  return res.json();
}
```

Funkcja

HTML

```
<img
  src={`http://localhost:3000/uploads/${photo.filename}`}
  width="200"
/>
```

POSTMAN


Home
Workspaces
API Network
Search Postman
⌘ K
Invite
Upgrade
Blog HZKWT's Workspace
New
Import
GET Get data
No environment
Collections
Environments
Flows
History
Search collections
My Collection
GET Get data
POST Post data
My Collection / Get data
Save
Share
GET
Enter URL or paste text
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Headers
6 hidden
Key
Value
Description
Bulk Edit
Presets
Key
Value
Description
Configure workspace sidebar

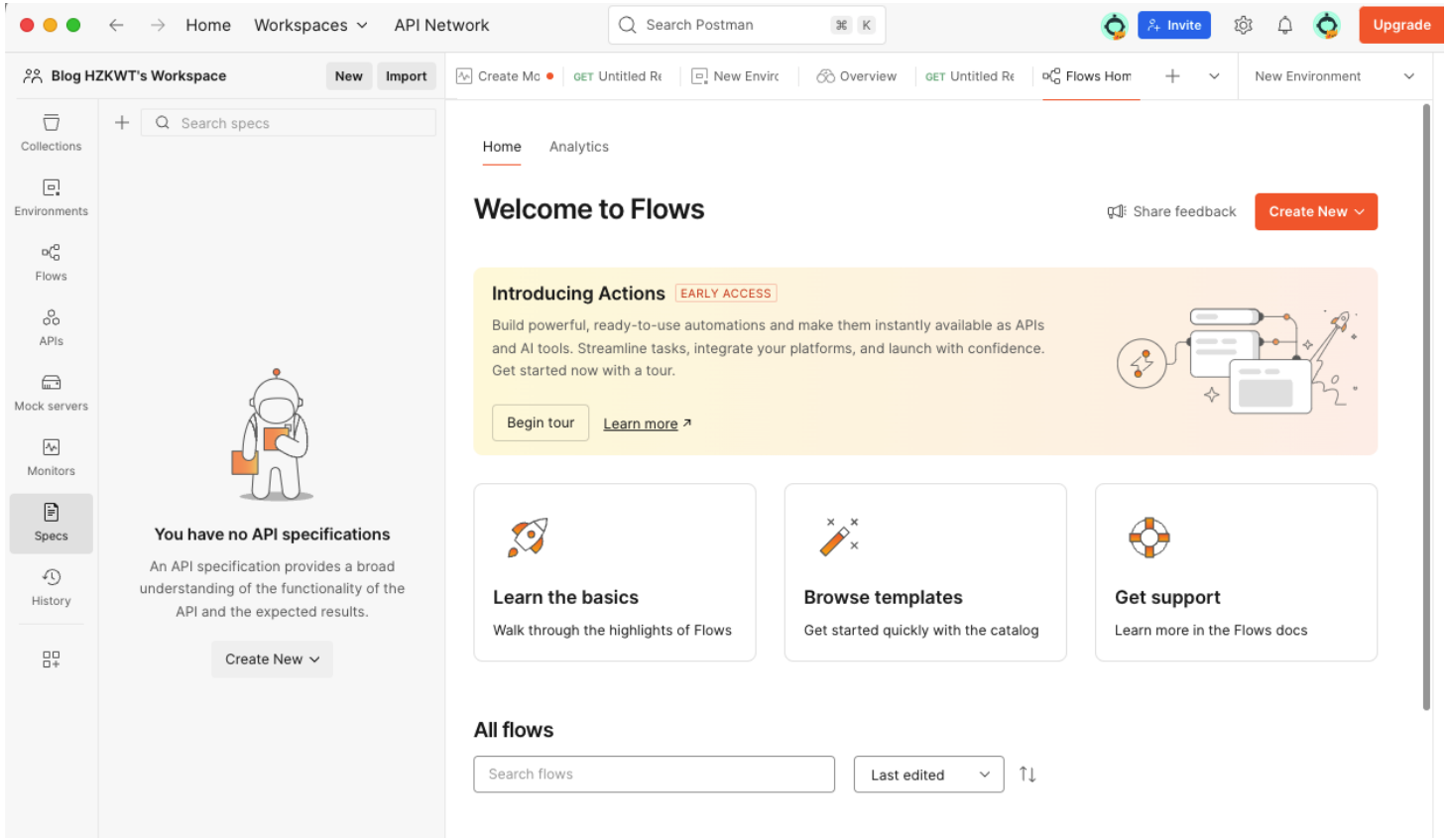
POSTMAN – przykłady zastosowań

To narzędzie do testowania aplikacji działających na przeglądarce.

Można sprawdzić szybkość odbierania i wysyłania danych.

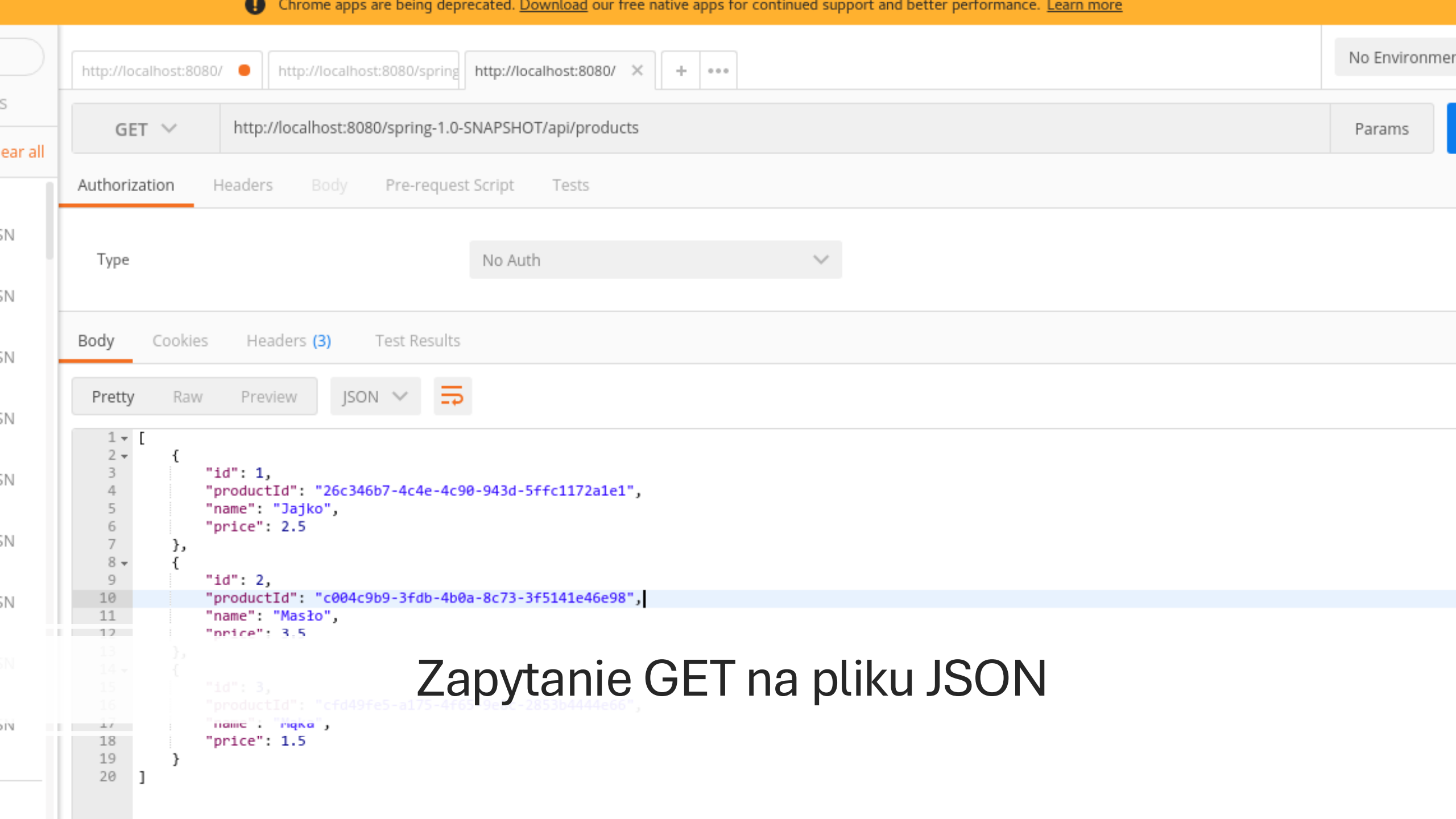
Można wygenerować „sztuczny ruch” na stronie i sprawdzić odporność na różnego rodzaju ataki.

Po każdym teście można wygenerować raport.



Kody błędów w POSTMAN

200 – OK,
201 – Created,
204 – No Content
400 – Złe zapytanie (Brakuje w zapytaniu wymaganych parametrów lub nie może być poprawnie odczytane)
401 – Niepowodzenie autoryzacji
403 – Dostęp zabroniony
404 – Nie znaleziono
405 – Niedozwolona metoda (Np. wysłanie pod dany adres metody POST, a nie GET)
500 – Błąd serwera
503 – Serwis niedostępny



Zapytanie GET na pliku JSON

Kompletny test API

Założenie testu: sprawdzenie poprawności dodania nowego użytkownika.

POST http://localhost:3000/api/users

Punkt końcowy

```
{
  "name": "Jan",
  "email": "jan@test.pl"
}
```

Dodajemy

```
{
  "id": 5,
  "name": "Jan",
  "email": "jan@test.pl"
}
```

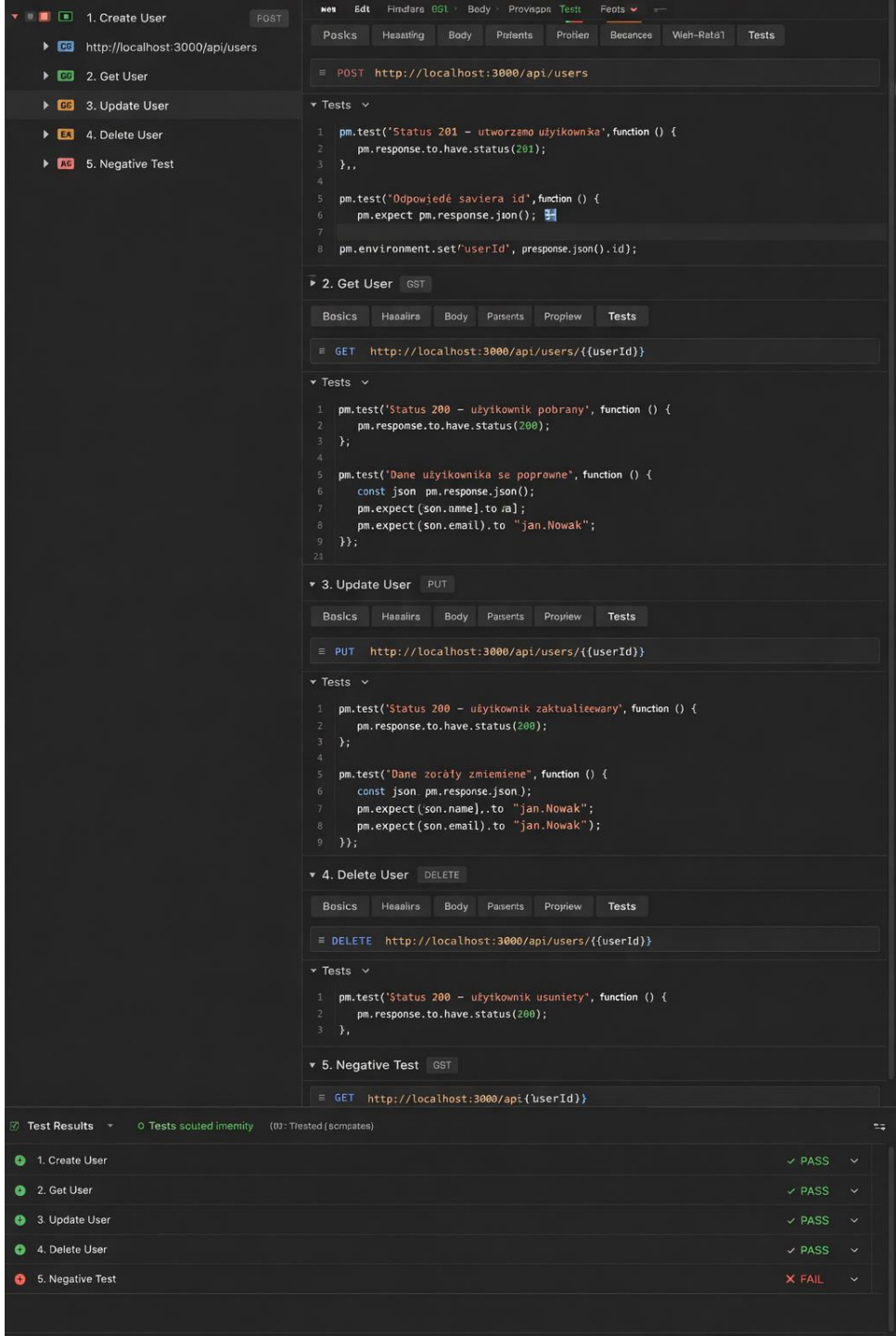
Oczekujemy

TEST

```
pm.test("Status odpowiedzi to 201", function () {
    pm.response.to.have.status(201);
});

pm.test("Odpowiedź jest w formacie JSON", function () {
    pm.response.to.be.json;
});

pm.test("Odpowiedź zawiera id użytkownika", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("id");
});

pm.test("Imię użytkownika jest poprawne", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.name).to.eql("Jan");
});

pm.test("Email użytkownika jest poprawny", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.email).to.eql("jan@test.pl");
});
```

Tabela testów do dokumentacji

Metoda	Endpoint	Opis	Oczekiwany wynik
POST	/api/users	Dodanie użytkownika	201 Created
GET	/api/users/{id}	Pobranie użytkownika	200 OK
PUT	/api/users/{id}	Aktualizacja danych	200 OK
DELETE	/api/users/{id}	Usunięcie użytkownika	200 OK
GET	/api/users/{id}	Pobranie po usunięciu	404 Not Found



Bootstrap w React - formatowanie

W przypadku korzystania z klas CSS:

npm install bootstrap - instalacja

import 'bootstrap/dist/css/bootstrap.min.css'; - import biblioteki

<button className="btn btn-primary">Kliknij mnie</button>. - zastosowanie w pliku projektu

Przy korzystaniu z komponentów reactowych:

npm install react-bootstrap bootstrap - instalacja

import 'bootstrap/dist/css/bootstrap.min.css'; - import biblioteki

import Button from 'react-bootstrap/Button'; - import komponentu

```
function App() {
  return <Button variant="success">OK</Button>;
}
```

Klasy Bootstrap – kontenery i layout

Klasa	Opis
`container`	Responsywny kontener z paddingiem
`container-fluid`	Kontener pełnej szerokości
`row`	Wiersz w systemie siatki
`col` / `col-4` / `col-md-6` / `col-lg-3`	Kolumna w systemie siatki
`d-flex`	Flexbox display
`flex-column` / `flex-row`	Kierunek flexa
`justify-content-center`	Wyśrodkowanie w poziomie
`align-items-center`	Wyśrodkowanie w pionie
`gap-2` / `gap-3` / `gap-4`	Odstępy między kolumnami/wierszami

Formularze

Klasa	Opis
`form-control`	Podstawowy input / textarea
`form-label`	Etykieta pola
`form-text`	Dodatkowy tekst pod polem (np. podpowiedź)
`form-check`	Kontener dla checkbox/radio
`form-check-input`	Input typu checkbox/radio
`form-check-label`	Label dla checkbox/radio
`is-invalid`	Czerwone podświetlenie błędu
`is-valid`	Zielone podświetlenie poprawnego wypełnienia

Przyciski i karty

Klasa	Opis
btn	Podstawowy przycisk
btn-primary / btn-secondary / btn-success / btn-danger / btn-warning / btn-info / btn-light/ btn-dark	Kolor przycisku
btn-outline-primary / btn-outline-success	Przycisk z obramowaniem
btn-lg / btn-sm	Duży / mały przycisk
btn-block	Pełna szerokość (Bootstrap 4) / w-100 w Bootstrap 5
disabled	Wyłączenie przycisku

Klasa	Opis
card	Podstawowy komponent karty
card-body	Sekcja treści karty
card-title	Tytuł karty
card-text	Tekst w karcie
card-header / card-footer	Nagłówek / stopka karty
card-img-top	Obrazek na górze karty
shadow / shadow-sm / shadow-lg	Cienie kart
rounded / rounded-lg / rounded-circle	Zaokrąglone rogi karty lub obrazów

Tekst i tła

Klasa	Opis
text-center / text-start / text-end	Wyrównanie tekstu
text-primary / text-secondary / text-success / text-danger / text-warning / text-info / text-light / text-dark	Kolory tekstu
fw-bold / fw-semibold / fw-normal / fw-light	Grubość tekstu
fst-italic / fst-normal	Kursywa / normalny font
text-truncate	Skracanie tekstu z "..."

Klasa	Opis
bg-primary / bg-secondary / bg-success / bg-danger / bg-warning / bg-info / bg-light / bg-dark	Kolor tła elementu
bg-transparent	Przezroczyste tło
border	Standardowa obwódka
border-primary / border-success / border-danger	Kolor obramowania
rounded / rounded-circle / rounded-pill	Zaokrąglenie rogów

Przykładowy prosty projekt

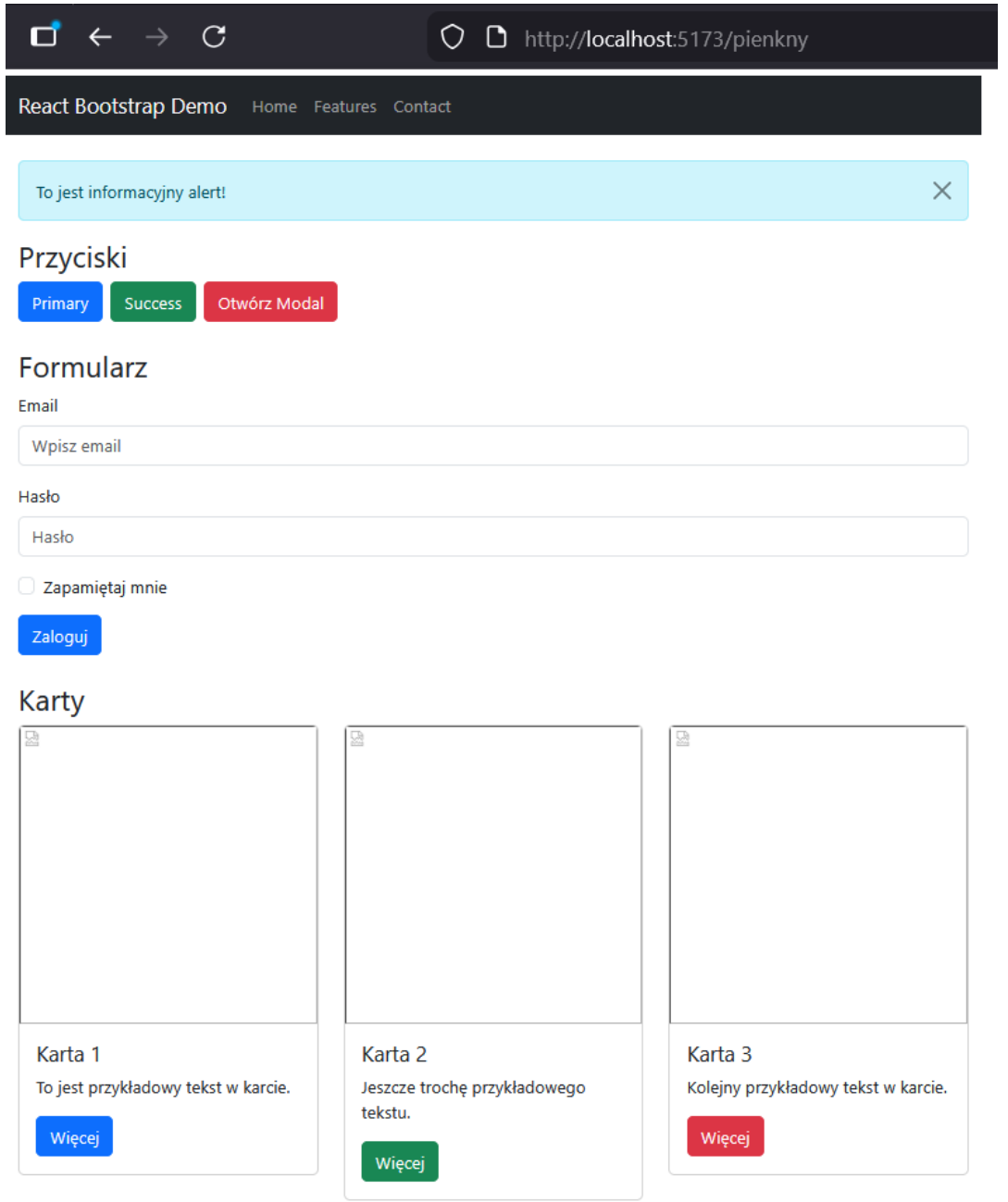
Utworzenie projektu:

npx create-react-app bootstrap-demo

cd bootstrap-demo

npm install bootstrap

npm start



Przykładowy prosty projekt

```
import React, { useState } from 'react';

import { Container, Row, Col, Button, Alert, Card, Form, Modal, Navbar, Nav } from 'react-bootstrap';


function App() {

  const [showModal, setShowModal] = useState(false);

  const [alertVisible, setAlertVisible] = useState(true);


  return (

    <>

      {/* Navbar */}

      <Navbar bg="dark" variant="dark" expand="lg" className="mb-4">

        <Container>

          <Navbar.Brand href="#">React Bootstrap Demo</Navbar.Brand>

          <Navbar.Toggle aria-controls="basic-navbar-nav" />

          <Navbar.Collapse id="basic-navbar-nav">

            <Nav
```

```
className="me-auto">

              <Nav.Link href="#">Home</Nav.Link>

              <Nav.Link href="#">Features</Nav.Link>

              <Nav.Link href="#">Contact</Nav.Link>

            </Nav>

          </Navbar.Collapse>

        </Container>

      </Navbar>


      <Container>

        {/* Alert */}

        {alertVisible && (

          <Alert variant="info" onClose={() => setAlertVisible(false)} dismissible>

            To jest informacyjny alert!

          </Alert>

        )}


        {/* Buttons */}

        <h3>Przyciski</h3>

        <Button variant="primary" className="me-2">Primary</Button>

        <Button variant="success" className="me-2">Success</Button>

        <Button variant="danger" onClick={() =>
```

```
setShowModal(true)}>Otwórz Modal</Button>


        {/* Modal */}

        <Modal show={showModal} onHide={() => setShowModal(false)}>

          <Modal.Header closeButton>

            <Modal.Title>Przykładowy Modal</Modal.Title>

          </Modal.Header>

          <Modal.Body>To jest treść modala.</Modal.Body>

          <Modal.Footer>

            <Button variant="secondary" onClick={() => setShowModal(false)}>Zamknij</Button>

            <Button variant="primary" onClick={() => setShowModal(false)}>Zapisz zmiany</Button>

          </Modal.Footer>

        </Modal>
```

Przykładowy

```
        {/* Forms */}
            <h3 className="mt-4">Formularz</h3>
            <Form>
                <Form.Group className="mb-3" controlId="formEmail">
                    <Form.Label>Email</Form.Label>
                    <Form.Control type="email" placeholder="Wpisz email" />
                </Form.Group>
                <Form.Group className="mb-3" controlId="formPassword">
                    <Form.Label>Hasło</Form.Label>
                    <Form.Control type="password" placeholder="Hasło" />
                </Form.Group>
                <Form.Check type="checkbox" label="Zapamiętaj mnie" className="mb-3" />
                <Button variant="primary" type="submit">Zaloguj</Button>
            </Form>

            {/* Cards */}
            <h3 className="mt-4">Karty</h3>
            <Row>
                <Col md={4} className="mb-3">
                    <Card>
                        <Card.Img variant="top" src="https://via.placeholder.com/150" />
                        <Card.Body>
                            <Card.Title>Karta 1</Card.Title>
                            <Card.Text>To jest przykładowy tekst w karcie.</Card.Text>
                            <Button variant="primary">Więcej</Button>
                        </Card.Body>
                    </Card>
                </Col>
                <Col md={4} className="mb-3">
                    <Card>
                        <Card.Img variant="top" src="https://via.placeholder.com/150" />
                        <Card.Body>
                            <Card.Title>Karta 2</Card.Title>
                            <Card.Text>Jeszcze trochę przykładowego tekstu.</Card.Text>
                            <Button variant="success">Więcej</Button>
                        </Card.Body>
                    </Card>
                </Col>
                <Col md={4} className="mb-3">
                    <Card>
                        <Card.Img variant="top" src="https://via.placeholder.com/150" />
                        <Card.Body>
                            <Card.Title>Karta 3</Card.Title>
                            <Card.Text>Kolejny przykładowy tekst w karcie.</Card.Text>
                            <Button variant="danger">Więcej</Button>
                        </Card.Body>
                    </Card>
                </Col>
            </Row>
        </Container>
    </>
  );
}

export default App;
```

Uwaga!

Ważne!

Pamiętaj aby aktualizować
i porównywać wersje.

Poprawna i terminowa aktualizacja pomaga wyświetlać wszystkie formatowania.

React	Vite	Bootstrap	Status
React 18.x	Vite 5.x	**Bootstrap 5.3.x**	✅ najlepsze
React 18.x	Vite 4.x	Bootstrap 5.2–5.3	✅ OK
React 17.x	Vite 4.x	Bootstrap 5.1–5.3	⚠️ starsze
React 16.x	Vite 3.x	Bootstrap 4.x	❌ niezalecane

Bootstrap+Vite

npm create vite@latest vite-bootstrap – instalacja
✓ React
✓ JavaScript
Następnie:
cd vite-bootstrap
npm install
npm install bootstrap
npm run dev

```
vite-bootstrap/
 ├─ src/
 │   ├─ App.jsx
 │   ├─ main.jsx
 │   └─ index.css
 ├─ index.html
 └─ package.json
```

Main.jsx:

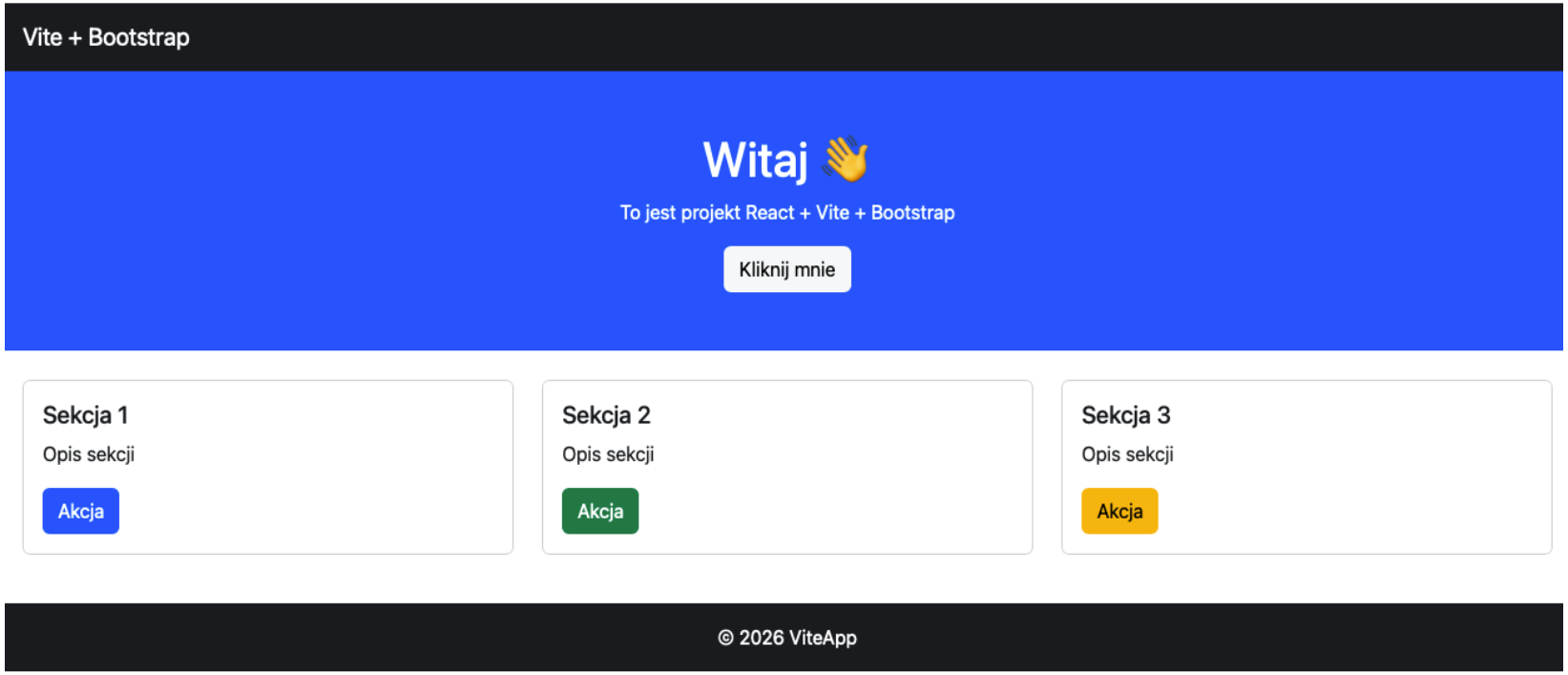
```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'

// Bootstrap
import 'bootstrap/dist/css/bootstrap.min.css'

ReactDOM.createRoot(document.getElementB
yId('root')).render(
 <React.StrictMode>
  <App />
 </React.StrictMode>
)
```

Bootstrap+Vite

```
function App() {
 return (
  <>

   {/* NAVBAR */}
   <nav className="navbar navbar-dark bg-dark">
    <div className="container">
     <span className="navbar-brand">Vite + Bootstrap</span>
    </div>
   </nav>

   {/* HERO */}
   <div className="bg-primary text-white text-center p-5">
    <h1>Witaj 👏</h1>
    <p>To jest projekt React + Vite + Bootstrap</p>
    <button className="btn btn-light">Kliknij mnie</button>
   </div>

   {/* KARTY */}
   <div className="container mt-4">
    <div className="row">

     <div className="col-md-4">
      <div className="card mb-3">
       <div className="card-body">
        <h5 className="card-title">Sekcja 1</h5>
        <p className="card-text">Opis sekcji</p>
        <button className="btn btn-primary">Akcja</button>
       </div>
      </div>
     </div>

     <div className="col-md-4">
      <div className="card mb-3">
       <div className="card-body">
        <h5 className="card-title">Sekcja 2</h5>
        <p className="card-text">Opis sekcji</p>
        <button className="btn btn-success">Akcja</button>
       </div>
      </div>
     </div>

     <div className="col-md-4">
      <div className="card mb-3">
       <div className="card-body">
        <h5 className="card-title">Sekcja 3</h5>
        <p className="card-text">Opis sekcji</p>
        <button className="btn btn-warning">Akcja</button>
       </div>
      </div>
     </div>

    </div>
   </div>

   {/* FOOTER */}
   <footer className="bg-dark text-white text-center p-3 mt-4">
    © 2026 ViteApp
   </footer>

  </>
 )
}

export default App
```

Bootstrap+Vite

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml"
href="/favicon.svg" />
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0" />
    <title>bajojajo</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module"
src="/src/main.jsx"></script>
  </body>
</html>
```



Tailwind

Tailwind CSS to w telegraficznym skrócie ogromny zbiór klas, które służą do stylowania elementów HTML.

Każda klasa ma jedno zadanie - np. klasa *.flex* po prostu dodaje *display: flex*, do elementu, a klasa *.text-orange-500* zmieni kolor tekstu na pomarańczowy.

Zapewne zastanawiasz się, czy nie jest to przypadkiem zakamuflowane pisanie styli “inline”.

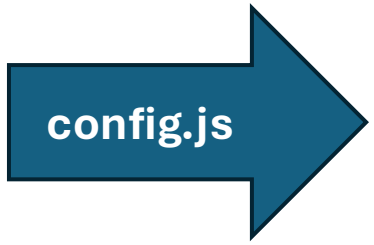
Może tak to wygląda, a mimo to Tailwind jest uznawany za jeden z najlepszych w swojej klasie.

Szybko okazuje się, że **takie podejście do stylowania oszczędza nam dużo samodzielnego pisania kodu!**

Tailwind – instalacja i konfiguracja

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p




```
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,jsx}"
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```


```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Najważniejsze klasy

Efekt	Klasa
Flex center	flex items-center justify-center
Gradient	bg-gradient-to-r from-blue-500 to-indigo-600
Zaokrąglenia	rounded-xl / rounded-2xl
Cień	shadow-lg / shadow-xl
Hover	hover:bg-indigo-700
Animacja	transition
Focus	focus:ring-2 focus:ring-indigo-500

Układy i kontenery

Klasa	Opis
min-h-screen	Minimalna wysokość równoważna wysokości ekranu
h-screen	Pełna wysokość ekranu
w-full	Szerokość 100%
max-w-sm / max-w-md / max-w-lg	Ograniczenie maksymalnej szerokości kontenera
p-4 / p-6 / p-8	Padding wokół elementu
m-4 / m-6 / m-8	Margin wokół elementu
space-y-4 / space-x-4	Odstęp między elementami w kolumnie lub w wierszu
flex	Flexbox
flex-col / flex-row	Kierunek flexa
items-center / justify-center	Wyśrodkowanie w osi cross/main
gap-4 / gap-6	Odstępy między elementami flex/grid

Tło i gradienty

Klasa	Opis
bg-white	Białe tło
bg-gray-100 / bg-gray-200	Jasne szare tło
bg-indigo-500 / bg-blue-500	Kolor tła Tailwind
bg-gradient-to-r	Gradient w prawo
from-blue-500	Początek gradientu
to-indigo-600	Koniec gradientu
hover:bg-indigo-700	Efekt hover na tle
bg-opacity-50	Przezroczystość tła

Obramowania i tekst

Klasa	Opis
border	Standardowa obwódka
border-2 / border-4	Grubość obramowania
border-red-500 / border-green-500	Kolor obramowania
rounded / rounded-md / rounded-lg / rounded-2xl	Zaokrąglone rogi
shadow / shadow-md / shadow-xl	Cienie elementu

Klasa	Opis
text-white / text-gray-700 / text-red-500	Kolory tekstu
text-sm / text-base / text-lg / text-xl / text-2xl / text-3xl	Rozmiar tekstu
font-bold / font-medium / font-light	Grubość fontu
text-center / text-left / text-right	Wyrównanie tekstu
uppercase / capitalize / lowercase	Transformacja tekstu
truncate	Skrócenie tekstu z "..." jeśli za długi

Interakcje

Klasa	Opis
`hover:bg-indigo-700`	Zmiana koloru tła na hover
`hover:text-white`	Zmiana koloru tekstu na hover
`focus:outline-none`	Usuwa domyślny focus outline
`focus:ring-2` / `focus:ring-4`	Pierścień przy focus
`focus:ring-indigo-500`	Kolor pierścienia przy focus
`transition` / `duration-200` / `ease-in-out`	Płynne przejścia przy hover/focus

App.jsx z formatowaniem

```jsx
import { useState } from "react";

function App() {
  const [login, setLogin] = useState("");
  const [password, setPassword] = useState("");
  const [message, setMessage] = useState("");
  const [error, setError] = useState("");

  const handleLogin = (e) => {
    e.preventDefault();
    setError("");
    setMessage("");

    // Walidacja
    if (!login || !password) {
      setError("Wypełnij wszystkie pola ❌");
      return;
    }

    // Przykładowa logika logowania
    if (login === "admin" && password === "1234") {
      setMessage("Zalogowano poprawnie ✅");
    } else {
      setError("Błędny login lub hasło ❌");
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gradient-to-r from-blue-500 to-indigo-600 p-4">
      <form
        onSubmit={handleLogin}
        className="bg-white p-8 rounded-2xl shadow-xl w-full max-w-sm space-y-4"
      >
        <h2 className="text-2xl font-bold text-center text-gray-700">
          Logowanie
        </h2>

        {/* Input Login */}
        <div>
          <label className="block mb-1 font-medium text-gray-600">Login</label>
          <input
            type="text"
            placeholder="Wpisz login"
            className="w-full px-4 py-2 border rounded-lg focus:outline-none focus:ring-2 focus:ring-indigo-500"
            value={login}
            onChange={(e) => setLogin(e.target.value)}
          />
        </div>

        {/* Input Hasło */}
        <div>
          <label className="block mb-1 font-medium text-gray-600">Hasło</label>
          <input
            type="password"
            placeholder="Wpisz hasło"
            className="w-full px-4 py-2 border rounded-lg focus:outline-none focus:ring-2 focus:ring-indigo-500"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
        </div>

        {/* Błąd */}
        {error && <p className="text-red-500 text-sm text-center">{error}</p>}

        {/* Sukces */}
        {message && <p className="text-green-500 text-sm text-center">{message}</p>}

        <button
          type="submit"
          className="w-full bg-indigo-600 text-white py-2 rounded-lg hover:bg-indigo-700 transition"
        >
          Zaloguj
        </button>
      </form>
    </div>
  );
}

export default App;
```

Efekt?

Prosty formularz z walidacją i formatowaniem w postaci zaokrąglonych pól i przycisku.