Wstęp teoretyczny (o formularzach)

Formularze są kluczowym elementem aplikacji internetowych React. Pozwalają użytkownikom do bezpośredniego wprowadzania i przesyłania danych w komponentach, począwszy od ekranu logowania po stronę płatności. Ponieważ większość aplikacji React to single page applications (SPA) lub aplikacje internetowe, w których nowe dane wyświetlają się dynamicznie. Informacje nie są przesyłane bezpośrednio z formularza na serwer. Zamiast tego operuje się tymi danymi po stronie klienta i przesyła się je lub wyświetla za pomocą dodatkowego kodu JavaScript.

0 krok – Tworzenie aplikacji

W tym kroku stworzymy szkielet aplikacji. Aby to zrobić używamy komendy:

`npx create-react-app form-tutorial`

Następnie tworzymy folder `components` w folderze `src` i wrzucamy tam nasze pliki `App.js` oraz `App.css`

Inforamcję dodatkowe:

… - przed i po blokach z kodem świadczy o tym, że jest to fragment kodu w pliku

X - na końcu linii z kodem mówi o tym, że linia ta została zmieniona lub dodana

1 krok – Tworzenie podstawowego formularza za pomocą JSX

W tym kroku stworzymy prosty formularz. Będzie to formularz do zakupu jabłek.

form-tutorial/src/components/App/App.js

```
import './App.css';

function App() {
  return (
    <div className='wrapper'>
      <h1>How About Them Apples</h1>
      <form>
```

```
          <label>
            <p>Name</p>
            <input name="name"/>
          </label>
        <button type='submit'>Submit</button>
      </form>
    </div>
  );
}

export default App;
```

Następnie dodajemy style css

form-tutorial/src/components/App/App.css

```
.wrapper {
  padding: 20px, 20px;
}

label {
  display: block;
  margin: 10px 0;
}
```

Jeśli teraz klikniemy na przycisk to strona się przeładuje. My jednak chcemy zapobiec takiemu zachowaniu i obsłużyć zdarzenie submit w środku naszego komponentu.

Aby to zrobić stworzymy event handler dla formularza. Stworzymy funkcje, w której zapobiegniemy domyślnemu zachowaniu oraz dodamy alert.

form-tutorial/src/components/App/App.js

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

function App() {
  const handleSubmit = event => {
    event.preventDefault();
    alert('You have subbmited the form.')
  }
  return (
    <div className='wrapper'>
      <h1>How About Them Apples</h1>
      <form onSubmit={handleSubmit}>
          <label>
```

```
            <p>Name</p>
            <input name="name"/>
          </label>
        <button type='submit'>Submit</button>
      </form>
    </div>
  );
}

export default App;
```

Teraz dodamy sobie opóźnienie, aby zasymulować API i dodamy hook, który pozwoli nam kontrolować czy czas już upłyną.

form-tutorial/src/components/App/App.js

```
import './App.css';
import React, {useState} from 'react';                                  X

function App() {
  const [submitting, setSubmitting] = useState(false);                  X
  const handleSubmit = event => {
    event.preventDefault();
    setSubmitting(true);                                                X

    setTimeout(() => {                                                  X
      setSubmitting(false);                                             X
    }, 3000)                                                            X
  }

  return (
    <div className='wrapper'>
      <h1>How About Them Apples</h1>
      {submitting &&                                                    X
      <div>Submitting Form..</div>}                                     X
      <form onSubmit={handleSubmit}>
          <label>
            <p>Name</p>
            <input name="name"/>
          </label>
        <button type='submit'>Submit</button>
      </form>
    </div>
  );
}

export default App;
```

2 krok – Zbieranie danych z formularza za pomocą niekontrolowanych komponentów.

W skrócie niekontrolowany komponent jest to komponent, który nie ma własności value ustawionej przez React.

Obecnie mamy formularz, który można przesłać. Jednak nie zbiera on danych. Będziemy musieli zatem stworzyć sposób na zarządzaniem naszym stanem. Do zarządzania naszym stanem posłużymy się hookiem useReducer.

form-tutorial/src/components/App/App.js

```
import './App.css';
import React, { useReducer, useState} from 'react';                    X

const formReducer = (state, event) => {                                X
  return {                                                             X
    ...state,                                                          X
    [event.name]: event.value                                          X
  }                                                                    X
}

function App() {
  const [formData, setFormData] = useReducer(formReducer, {});         X
  const [submitting, setSubmitting] = useState(false);
  const handleSubmit = event => {
    event.preventDefault();
    setSubmitting(true);

    setTimeout(() => {
      setSubmitting(false);
    }, 3000)
  }

  const handleChange = event => {                                      X
    setFormData({                                                      X
      name: event.target.name,                                         X
      value: event.target.value,                                       X
    })
  }

  return (
    <div className='wrapper'>
      <h1>How About Them Apples</h1>
      {submitting &&
      <div>Submitting Form..</div>}
      <form onSubmit={handleSubmit}>
          <label>
```

```
            <p>Name</p>
            <input name="name" onChange={handleChange}/>
          </label>
        <button type='submit'>Submit</button>
      </form>
    </div>
  );
}

export default App;
```

Stworzyliśmy sobie Reducer, który zapisuje zmienne do stanu dodatkowo stworzyliśmy handleChange, aby zdobywać zmienne z inputów i przekazywać je do stanu.

Teraz dodamy sobie listę ze zmiennymi, które otrzymujemy.

form-tutorial/src/components/App/App.js

...

```
  {submitting &&
      <div>You are subbmiting the following:
        <ul>
          {Object.entries(formData).map(([name, value]) => (
            <li key={name}><strong>{name}</strong>:{value.toString()}</li>

          ))}
        </ul>
      </div>}
```

...

Za pomocą Object.entries zamieniamy nasze dane w tablice i używamy funkcji map, aby wypisać wszystkie zmienne w tej tablicy. W naszym przypadku są to tylko zmienne name.

Teraz dodamy sobie więcej pól w naszym formularzu

form-tutorial/src/components/App/App.js

...

```
<form onSubmit={handleSubmit}>
          <label>
```

```
          <p>Name</p>
          <input name="name" onChange={handleChange}/>
        </label>
        <label>
          <p>Apples</p>
          <select name="apple" onChange={handleChange}>
            <option value="">--Please choose an option--</option>
            <option value="fuji">Fuji</option>
            <option value="jonathan">Jonathan</option>
            <option value="honey-crisp">Honey Crisp</option>
          </select>
        </label>
        <label>
          <p>Count</p>
          <input type="number" name="count" onChange={handleChange}/>
        </label>
        <label>
          <p>Gift Wrap</p>
          <input type="checkbox" name="gift-wrap" onChange={handleChange}/>
        </label>
      <button type='submit'>Submit</button>
    </form>
```

...

Teraz musimy pochylić się jeszcze nad checkboxem, ponieważ w tym momencie zawsze będzie przekazywał wartość on. Będziemy musieli użyć własności checked. W funkcji handleChange sprawdzimy czy przekazany obiekt jest checkboxem, a jeśli będzie odwołamy się do własności checked.

form-tutorial/src/components/App/App.js

...

```
const handleChange = event => {
  const isCheckbox = event.target.type === 'checkbox';
  setFormData({
    name: event.target.name,
    value: isCheckbox ? event.target.checked : event.target.value,
  })
}
```

...

3 krok – Zamiana danych formularza z użyciem kontrolowanych komponentów.

Dzięki niekontrolowanym komponentom nie trzeba martwić się o synchronizację danych. Ale jest wiele sytuacji, w których będziesz musiał zarówno czytać jaki zapisywać do komponentu wejściowego i aby to zrobić potrzebna jest własność wartość (value), a zatem komponenty kontrolowane.

Dodajmy zatem własności value do naszych inputów.

form-tutorial/src/components/App/App.js

...

```
<div className='wrapper'>
      <h1>How About Them Apples</h1>
      {submitting &&
      <div>You are subbmiting the following:
        <ul>
          {Object.entries(formData).map(([name, value]) => (
            <li key={name}><strong>{name}</strong>:{value.toString()}</li>
          ))}
        </ul>
      </div>}
      <form onSubmit={handleSubmit}>
          <label>
            <p>Name</p>
            <input name="name" onChange={handleChange}
              value={formData.name || ''}/>                                    X
          </label>
          <label>
            <p>Apples</p>
            <select name="apple" onChange={handleChange}
              value={formData.apple || ''}>                                    X
              <option value="">--Please choose an option--</option>
              <option value="fuji">Fuji</option>
              <option value="jonathan">Jonathan</option>
              <option value="honey-crisp">Honey Crisp</option>
            </select>
          </label>
          <label>
            <p>Count</p>
            <input type="number" name="count" onChange={handleChange}
              value={formData.count || ''}/>                                   X
          </label>
          <label>
            <p>Gift Wrap</p>
            <input type="checkbox" name="gift-wrap" onChange={handleChange}
              checked={formData['gift-wrap'] || false}/>                       X
          </label>
        <button type='submit'>Submit</button>
      </form>
```

```
    </div>
```

...

Ponownie nasz checkbox jest nieco inny. Zamiast ustawiać wartość ustawiamy własność checked

Możemy ustawiać domyślne wartości dla naszych inputów. Zatem dodamy teraz domyślną wartość dla ilości w naszym formularzu.

form-tutorial/src/components/App/App.js

...

```
function App() {
  const [formData, setFormData] = useReducer(formReducer, {
    count: 100,
  });
```

...

Przydałoby się jeszcze resetowanie pól formularza po jego przesłaniu. Dodajmy więc resetowanie danych w inputach po kliknięciu submit.

form-tutorial/src/components/App/App.js

...

```
const formReducer = (state, event) => {
  if(event.reset){
    return {
      apple:  '',
      count: 0,
      name: '',
      'gift-wrap': false,
    }
  }
  return {
    ...state,
    [event.name]: event.value
  }
}

function App() {
  const [formData, setFormData] = useReducer(formReducer, {
    count: 100,
  });
  const [submitting, setSubmitting] = useState(false);
  const handleSubmit = event => {
    event.preventDefault();
    setSubmitting(true);
```

```
    setTimeout(() => {
      setSubmitting(false);
      setFormData({
        reset: true
      })
    }, 3000)
  }
```

...

Teraz nasze komponenty są już komponentami kontrolowanymi.

4 krok – Dynamiczne aktualizowanie właściwości formularza

W tym ostatnim już kroku ustawimy pola na nieaktywne podczas symulacji wysyłania naszych danych.

form-tutorial/src/components/App/App.js

```
import './App.css';
import React, { useReducer, useState} from 'react';

const formReducer = (state, event) => {
  if(event.reset){
    return {
      apple:  '',
      count: 0,
      name: '',
      'gift-wrap': false,
    }
  }
  return {
    ...state,
    [event.name]: event.value
  }
}

function App() {
  const [formData, setFormData] = useReducer(formReducer, {
    count: 100,
  });
  const [submitting, setSubmitting] = useState(false);
  const handleSubmit = event => {
    event.preventDefault();
    setSubmitting(true);

    setTimeout(() => {
      setSubmitting(false);
      setFormData({
        reset: true
```

```
    })
  }, 3000)
}

const handleChange = event => {
  const isCheckbox = event.target.type === 'checkbox';
  setFormData({
    name: event.target.name,
    value: isCheckbox ? event.target.checked : event.target.value,
  })
}

return (
  <div className='wrapper'>
    <h1>How About Them Apples</h1>
    {submitting &&
    <div>You are subbmiting the following:
      <ul>
        {Object.entries(formData).map(([name, value]) => (
          <li key={name}><strong>{name}</strong>:{value.toString()}</li>
        ))}
      </ul>
    </div>}
    <form onSubmit={handleSubmit}>
        <label>
          <p>Name</p>
          <input disabled={submitting} name="name"
            onChange={handleChange} value={formData.name || ''}/>
        </label>
        <label>
          <p>Apples</p>
          <select disabled={submitting} name="apple"
            onChange={handleChange} value={formData.apple || ''}>
            <option value="">--Please choose an option--</option>
            <option value="fuji">Fuji</option>
            <option value="jonathan">Jonathan</option>
            <option value="honey-crisp">Honey Crisp</option>
          </select>
        </label>
        <label>
          <p>Count</p>
          <input disabled={submitting} type="number"
            name="count" onChange={handleChange}
            value={formData.count || ''}/>
        </label>
        <label>
          <p>Gift Wrap</p>
          <input disabled={submitting} type="checkbox"
            name="gift-wrap" onChange={handleChange}
```

```
              checked={formData['gift-wrap'] || false}/>
          </label>
        <button type='submit' disabled={submitting}>Submit</button>          X
      </form>
    </div>
  );
}

export default App;
```

W ten sposób udało się nam ukończyć prosty formularz w React.