ToiletPaper #138

React-Formulare ohne State

Autor: Santo Pfingsten / Software Engineer / Standort Leipzig

✕ Problem

Die oft gelehrte Methode, ein Formular zu erstellen, ist mit State. Mit React Hooks sieht das ganze wie folgt aus:

```
export const RegistrationForm = ({ submitForm }) => {
    const [name, setName] = useState('');
    const [email, setEmail] = useState('');
    return (
        <form onSubmit={() => submitForm(name, email)}>
            <label>
                <b>Name: </b>
                <input value={name} onChange={(e) => setName(e.currentTarget.value)} />
            </label>
            <label>
                <b>E-Mail: </b>
                <input value={email} onChange={(e) => setEmail(e.currentTarget.value)} />
            </label>
            <button type="submit">Submit</button>
        </form>
    );
};
```

Ein Formular auf diese Art zu erstellen, hat sich quasi in die React-Community eingebrannt. Doch solche Formulare bestehen häufig aus mehr als nur 2 Eingabefeldern und wer genau hinschaut, merkt, dass für jedes weitere Eingabefeld hier mindestens 2 neue Zeilen hinzukommen. Stellt man sich ein Formular mit 30+ Einträgen vor, so wird es langsam unübersichtlich. Zudem wird bei jedem Tastendruck die Funktion neu aufgerufen, was ggf. auch zu Performanceproblemen führen kann.

✓ Lösung

Sofern man die Formularinhalte nur zum Abschicken benötigt (und z. B. Validierung nur serverseitig durchführt), kann man den State auch weglassen. Auf die Werte kann man dann mit der `FormData` Klasse zugreifen. `FormData` benötigt einzig das `form` Element als Konstruktorargument und kann entweder direkt als `body` Attribut an einen `fetch` Aufruf übergeben werden oder man kann über die enthaltenen Werte iterieren.

```
export const RegistrationForm = ({ submitForm }) => (
    <form onSubmit={(e) => submitForm(new FormData(e.currentTarget))}>
        <label><b>Name: </b><input name="name" /></label>
        <label><b>E-Mail: </b><input name="email" /></label>
        <button type="submit">Submit</button>
    </form>
);
function submitFormExample(formData: FormData) {
    formData.forEach((value, key) => console.log(key, value));
    for(const [key, value] of formData) { console.log(key, value); }
    fetch("/api/register", { body: formData, ... }).then(...);
}
```

Durch das Hinzufügen zusätzlicher `name` Attribute kann man diese im `FormData` Objekt identifizieren. Sie erscheinen dann im `FormData` als `key`.

+ Weiterführende Aspekte

- Dokumentation zu FormData: https://developer.mozilla.org/en-US/docs/Web/API/FormData