React Grundlagen

Lernziele

- Verstehen, was React ist und warum es verwendet wird
- Verstehen von JSX und Unterschiede zu HTML
- Verstehen des deklarativen Ansatzes von React
- Erstellen von React-Komponenten
- Verstehen des Renderns mit React
- Kenntnis des React-Ökosystems
- Was npm ist und wie es verwendet wird
- Was Pakete sind und wie das npm-Ökosystem funktioniert
- Die grundlegende Anatomie eines npm-Pakets
- Wie semantische Versionskontrolle funktioniert

Was ist React und warum nutzen wir es?

React ist eine JavaScript-Bibliothek, die darauf abzielt, das Leben der Entwickler zu erleichtern: In den meisten Fällen müssen Sie nicht direkt mit der DOM-API (z. B. `createElement`) arbeiten. Sie schreiben einfach einfacheren (deklarativen) Code, der beschreibt, wie die Benutzeroberfläche aussehen soll, und React kümmert sich im Hintergrund um das DOM.

Um deklarativen Code für React zu schreiben, verwenden Sie JSX.

Verwendung von JSX

JSX ist eine Syntaxerweiterung für JavaScript. JSX ist weder ein String noch HTML, wie wir es kennen. JSX-Ausdrücke können überall dort verwendet werden, wo ein JavaScript-Ausdruck verwendet werden kann.

```
const element = <p>Some Text</p>;
```

Wir verwenden JSX, um React-Elemente zu erstellen. React-Elemente sind ein Zwischenformat, das React während des Renderprozesses in DOM-Elemente umwandelt. Dies ermöglicht es uns, unsere Benutzeroberfläche deklarativ mit JSX zu beschreiben.

Erstellen von Elementen

Ähnlich wie in HTML werden JSX-Elemente mit Öffnungs- und Schließ-Tags beschrieben. Das Öffnungstag enthält den Tag-Namen oder den Komponententyp (see Using Components) und alle Attribute. Das Schließ-Tag enthält denselben Tag-Namen oder denselben Komponententyp wie das Öffnungstag und nichts anderes. Die Kinder des Elements werden zwischen dem Öffnungs- und dem Schließ-Tag platziert. Wenn das Element keine Kinder hat, kann das Schließ-Tag weggelassen werden, und das Element ist selbstschließend.

```
// Element mit Kindern
//
//              Öffnungstag         Kinder
//              |  Attribute         |        Schließ-Tag
//              |  |                 |        |
const element = <p className="text">Some Text</p>;
//               | |         |                 |
//               | |         Attributwert      |
//               | Attributname                |
//              Tagname oder Komponententyp ---+

// Selbstschließendes Element
//
//            Selbstschließender Tag   Slash zeigt Selbstschließen an
//            |      Attribute         |
//            |      |                 |
const input = <input type="text" />;
//             |     |    |
//             |     |    Attributwert
//             |     Attributname
//            Tagname oder Komponententyp
```

> 💡 Elemente, die in HTML keine Schließ-Tags unterstützen wie `<br>` oder `<input>`, müssen in JSX selbstschließend sein (wie `<br />` oder `<input type="text" />`).

> 💡 Im Gegensatz zu HTML, das tolerant gegenüber fehlenden Schließ-Tags ist, ist JSX das nicht. Wenn Sie vergessen, ein Tag zu schließen, erhalten Sie einen Fehler.

Verwendung von Komponenten

Um ein Element aus einer component zu erstellen, können wir einfach auf den Funktionsnamen in JSX verweisen und es wie jede eingebaute Komponente behandeln:

```
const element = <MyComponent />;
```

Was Attribute und Kinder betrifft, funktioniert das Erstellen von Elementen aus Komponententypen genauso wie bei jedem (HTML) Tag-Namen.

> 💡 JSX unterscheidet zwischen eingebauten (HTML) Tag-Namen und Komponenten, indem es sich das erste Zeichen im JSX-Tag ansieht. Wenn es sich um Kleinbuchstaben handelt, wird es als eingebauter Tag-Name behandelt, wenn es sich um Großbuchstaben handelt, wird nach einer definierten JavaScript-Funktion mit diesem Namen gesucht. Deshalb ist es wichtig, PascalCase für Komponenten-Namen zu verwenden.

> 📙 Lesen Sie mehr über **Markup mit JSX schreiben** in der React-Dokumentation.

Attribute

Attribute für eingebaute HTML-Elemente verwenden JavaScript-zentrierte Namen aus der DOM-API. In den meisten Fällen sind die Namen die gleichen wie in HTML, aber es gibt einige Ausnahmen. Zum Beispiel wird das `class`-Attribut aus HTML in JSX als `className` bezeichnet.

Das Übergeben von String-Werten an Attribute erfolgt durch die Verwendung von Anführungszeichen. Um JavaScript-Ausdrücke zu übergeben, verwenden Sie geschweifte Klammern.

```
const element = <p className="text">Some Text</p>;

const myValue = "This is a string";
const input = <input type="text" value={myValue} minLength={5} />;
```

Verschachteln von Elementen

React-Elemente können auf die gleiche Weise verschachtelt werden, wie wir unsere HTML-Elemente verschachtelt haben.

```
const element = (
  <div>
    <p>Some Text</p>
    <p>Some more Text</p>
  </div>
);
```

> 💡 Mehrzeilige JSX-Ausdrücke werden in Klammern gesetzt, um sie lesbarer zu machen. Keine Sorge: Prettier kümmert sich darum für Sie.

Interpolation von Ausdrücken

Wir können jeden JavaScript-Ausdruck in JSX verwenden, indem wir ihn in geschweifte Klammern setzen. Dies wird als Interpolation bezeichnet. Es ist ähnlich wie die String-Interpolation in JavaScript-Template-Strings.

```
const name = "Pawtricia";
const element = <p>My cat's name is {name}</p>;
```

```
const a = 5;
const b = 10;

const element = (
  <p>
    {a} + {b} = {a + b}
  </p>
);
```

> 💡 Sie können nur Ausdrücke in JSX verwenden. Anweisungen wie if oder for sind nicht erlaubt.

> 💡 Um zu lernen, wie man JavaScript-Ausdrücke in JSX-Attributen interpoliert, siehe den Abschnitt Attribute.

> 📙 Lesen Sie mehr über JavaScript in JSX mit geschweiften Klammern in der React-Dokumentation.

React-Komponenten

React-Anwendungen werden mit Komponenten erstellt. Eine Komponente ist ein unabhängiges und wiederverwendbares Stück der Benutzeroberfläche, das seine eigene Struktur, Logik und möglicherweise auch Styling enthält.

React-Komponenten sind JavaScript-Funktionen, die React-Elemente zurückgeben. Diese Elemente werden dann während des Renderprozesses von React in DOM-Elemente umgewandelt.

Um eine React-Komponente zu erstellen, schreiben wir eine benannte Funktion (unter Verwendung von PascalCase) und lassen sie die gewünschten Elemente mit JSX zurückgeben.

```
function MyButton() {
  return (
    <button type="button" className="default-button">
      I'm a button
    </button>
  );
}
```

Dies ist ein sehr mächtiges Konzept, da es uns ermöglicht, dieselbe Komponente an mehreren Stellen in unserer Anwendung wiederzuverwenden.

> 💡 Siehe Verwendung von Komponenten für weitere Informationen zur Verwendung von Komponenten in JSX.

> 💡 Es gibt fast keine Einschränkungen, wie 'klein' eine Komponente sein kann (z. B. ein Button) oder wie 'groß' (z. B. eine ganze Seite).

> 📙 Lesen Sie über **Ihre erste Komponente in der React-Dokumentation.

> **Hinweis**: *Exportieren der Komponente* und *Verschachteln und Organisieren von Komponenten* sind für diese erste Sitzung nicht relevant.

Imperative vs. Deklarative Programmierung

Der Hauptunterschied zwischen imperativem und deklarativem Code besteht darin, dass imperativer Code beschreibt, *wie* etwas gebaut werden soll, während deklarativer Code beschreibt, *was* gebaut werden muss.

> 💡 Stellen Sie sich vor, Sie bauen einen Hocker. Imperativer "Code" würde die Schritte beschreiben, die erforderlich sind, um den Hocker zu bauen. Deklarativer "Code" würde den Hocker selbst beschreiben.

> Imperativ:

> - 4 Holzleisten nehmen
> - 1 Holzbrett nehmen
> - 4 Schrauben nehmen
> - Einen Schraubendreher nehmen
> - Die Leisten senkrecht unter das Brett schrauben
> - Ihre Arbeit so positionieren, dass das Brett oben ist

> Deklarativ:

> - Ein Hocker mit 4 Beinen und einer Sitzfläche, der aufrecht steht

In der imperativen Programmierung führt Ihr Code eine Reihe von Aktionen aus.

In der deklarativen Programmierung beschreibt Ihr Code ein gewünschtes Ergebnis.

Die Art und Weise, wie wir JavaScript in diesem Kurs bisher verwendet haben, war überwiegend imperativ. Wir haben beschrieben, was getan werden muss, um ein bestimmtes Ergebnis zu erzielen.

```
const p = document.createElement("p");
p.classList.add("introText");
p.textContent = "Hello World!";
rootElement.append(p);
```

Jetzt erlaubt es uns React, JavaScript auf deklarative Weise zu verwenden. Wir beschreiben React, was wir wollen, und React kümmert sich darum, das DOM gemäß unserer Beschreibung zu aktualisieren.

```
root.render(<p className="introText">Hello World!</p>);
// React könnte dies so interpretieren, dass es Folgendes tut:
// const p = document.createElement("p");
// p.classList.add("introText");
// p.textContent = "Hello World!";
// rootElement.append(p);
// …
```

Wie React rendert

React muss wissen, wo die von ihm erstellten Elemente gerendert werden sollen. Wir wählen das DOM-Element aus, in das wir rendern möchten, indem wir `document.querySelector()` verwenden. Dann erstellen wir ein React-Root-Objekt. Das Root-Objekt hat eine `render()`-Methode, die wir verwenden können, um React-Elemente in das DOM zu rendern.

HTML

```
<div id="root"></div>
```

Javascript

```
const rootElement = document.querySelector("#root");
const root = ReactDOM.createRoot(rootElement);
root.render(<h1>Hello, world</h1>);
```

Wahrscheinlich müssen Sie diesen Code nie selbst schreiben, da er bereits in allen Vorlagen und Startern enthalten ist. In der Praxis sieht es normalerweise so aus:

```
const rootElement = document.getElementById("root");
const root = ReactDOM.createRoot();

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Hier haben wir ein importiertes `<App />`-Element, das in `<React.StrictMode>` eingebettet ist.

> 💡 `StrictMode` richtet React so ein, dass es im strengen Modus läuft. Im strengen Modus weist React auf potenzielle Probleme in einer Anwendung hin.

React arbeitet intelligent, nicht hart

React aktualisiert nur die DOM-Elemente, die sich im Vergleich zum letzten Rendern geändert haben. Dies ist sehr effizient und sorgt für eine großartige Benutzererfahrung (der Fokus bleibt konsistent, Eingaben behalten ihre Werte usw.) sowie eine großartige Entwicklererfahrung (deklarativer Code ist viel einfacher zu verstehen).

Gut zu wissen: React, JSX, Transpilers und Bundlers

Da JSX keine standardisierte JavaScript-Syntax ist, müssen wir einen Transpiler (ein Tool, das eine Variante einer Sprache in eine andere übersetzt) verwenden, um es in standardmäßiges JavaScript zu transformieren, das vom Browser verstanden werden kann.

Ein Bundler ist ein Tool, das alle Dateien unseres Codes in eine einzige Datei kombiniert, die wir in unserem HTML einbinden können. Der Bundler kümmert sich auch darum, den Transpiler bei Bedarf auszuführen.

Der Bundler erstellt einen Entwicklungsserver, wenn wir `npm run start` lokal ausführen.

> 💡 Vielleicht haben Sie bemerkt, dass wir in den Aufgaben eine `import`-Anweisung verwenden, um `.css`-Dateien in unsere JavaScript-Dateien zu importieren. Dies ist kein Standard-JavaScript-Feature, wird jedoch vom Bundler unterstützt. Eine CSS-Import-Anweisung wird automatisch in ein `<link>`-Element im HTML umgewandelt.

```
import "./styles.css";
```

---

npm

Es ist ein Paket-Registry, das wie ein App-Store für Ihr Projekt funktioniert.

---

package.json

Pakete, die in Ihr Projekt installiert werden, werden als Abhängigkeiten bezeichnet. Sie werden in einer `package.json`-Datei im Projektstamm gespeichert. Die `package.json`-Datei enthält auch Informationen über Ihr Projekt.

Eine `package.json` könnte etwa so aussehen:

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "A description of my app",
  "scripts": {
    "test": "npm run …"
  },
  "author": "Alex Spiced",
  "license": "UNLICENSED",
  "dependencies": {
    "my-dependency": "^10.4.1",
    "my-other-dependency": "^2.0.0"
  },
  "devDependencies": {
    "my-dev-dependency": "^8.0.105",
    "my-other-dev-dependency": "^0.1.6"
  }
}
```

- `dependencies` sind Pakete, von denen Ihr Anwendungscode direkt abhängt, wie Bibliotheken oder Frameworks.
- `devDependencies` sind Pakete, die Ihnen bei der Entwicklung Ihrer Anwendung helfen, wie Linter oder Build-Tools.

Installieren von Abhängigkeiten

`dependencies` und `devDependencies` in der `package.json` können durch Ausführen von `npm install` (oder einfach `npm i` für kurz) installiert werden.

> 💡 Vergessen Sie nicht, `npm install` auszuführen, nachdem Sie ein neues Repository geklont haben, das eine `package.json`-Datei enthält.

Beim Installieren erstellt npm einen `node_modules`-Ordner und eine `package-lock.json`-Datei.

- `node_modules` muss immer in Ihrer `.gitignore`-Datei enthalten sein und darf nicht in Ihr Repository eingecheckt werden!
- `package-lock.json` sollte in Ihr Repository eingecheckt werden.

Semantische Versionierung

Eine semantische Version wird aktualisiert, wann immer sich ein Paket ändert und eine neue Version veröffentlicht wird.

Sie folgt diesem Schema: **`Major.Minor.Patch`** (z. B. 1.2.3)

- **`Major`** → Hauptversion, ändert sich, wenn sich die öffentliche API eines Pakets ändert (Breaking Change)
- **`Minor`** → Nebenversion, ändert sich, wenn neue Funktionen hinzugefügt werden
- **`Patch`** → Patch-Version, ändert sich, wenn Fehler behoben werden

Beim Definieren von Abhängigkeiten in `package.json verwendet npm Versionsbereiche, um festzulegen, welche Version eines Pakets installiert werden soll. npm installiert immer die neueste Version des Pakets, die noch mit der Bereichsbeschreibung übereinstimmt.

- `^` (z. B. `"^10.4.1"`) → Neuere Nebenupdates und Patches können installiert werden, aber Hauptupdates nicht. (Hier würde Version `10.5.6` installiert, aber nicht `11.0.0`)
- `~` (z. B. `"~10.4.1"`) → Neuere Patches können installiert werden, Neben- und Hauptupdates nicht. (Hier würde Version `10.4.8` installiert, aber nicht `10.5.0`)
- `>` (z. B. `">10.4.1"`) → Jede neuere Version wird installiert. (Hier würde jede Version neuer als `10.4.1` installiert)

Versionsbereiche, die mit `^` beschrieben werden, sind bei weitem die häufigste Wahl, da sie in der Regel sicher sind und Ihre Anwendung nicht beschädigen.

Projektstrukturierung mit `Create React App``

Die Projektstrukturierung ist der Prozess des Erstellens eines neuen Projekts. Sie verwenden das Create React App Tool, um ein neues React-Projekt automatisch zu erstellen.

> 💡 Prinzipiell könnten Sie ein neues React-Projekt von Grund auf neu erstellen. Dies wäre jedoch sehr arbeitsaufwendig, und wir müssten viele Dinge selbst einrichten. Beispielsweise müssten Sie einen Entwicklungsserver, einen Build-Prozess und einen Test-Runner einrichten. Sie müssten auch einen Modul-Bundler und einen Transpiler konfigurieren. Das ist viel Arbeit, und Sie müssten es jedes Mal tun, wenn Sie ein neues Projekt erstellen möchten.

> 💡 Create React App funktioniert übrigens ziemlich ähnlich wie das `ghcd`-Tool, das Sie wahrscheinlich bereits verwendet haben.

Resources

- What is React: A Visual Introduction For Beginners on learnreact.design

- Writing Markup with JSX in the React Docs
- JavaScript in JSX with Curly Braces in the React Docs
- Your First Component in the React Docs
- Difference between a Framework and a Library on freecodecamp

About npm

- npm website
- package.json specification
- npm install documentation
- Semantic Versioning specification