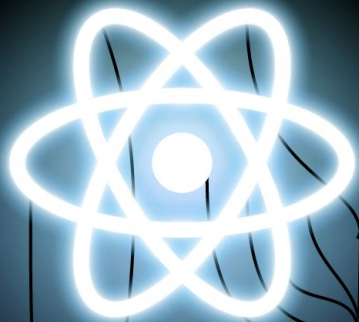


# React: Single Page Apps

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

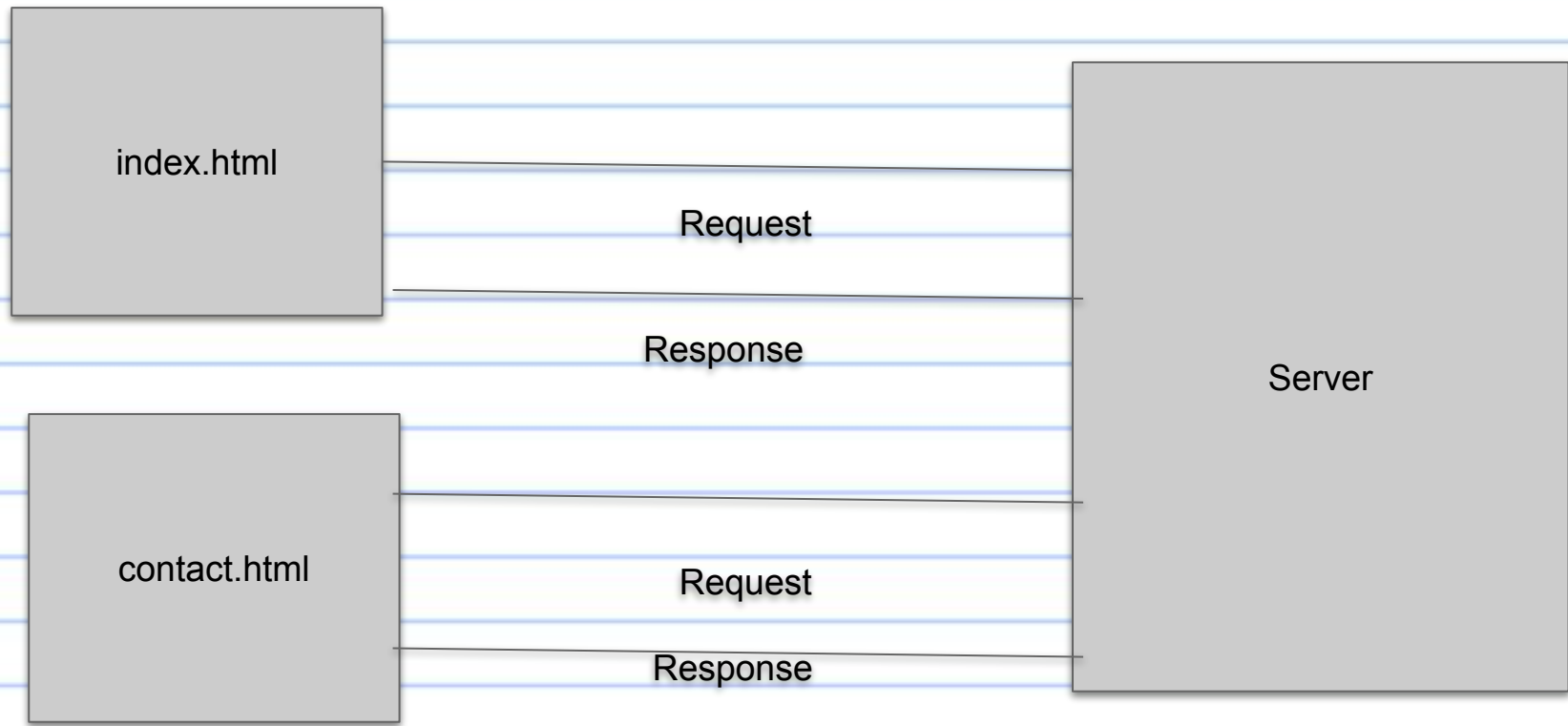
# React: Single Page Apps



- React Apps sind typischerweise SPA's
- Es wird immer nur EINE HTML Seite dem Browser vom Server gesendet
- React kontrolliert was der Nutzer auf der Seite sieht

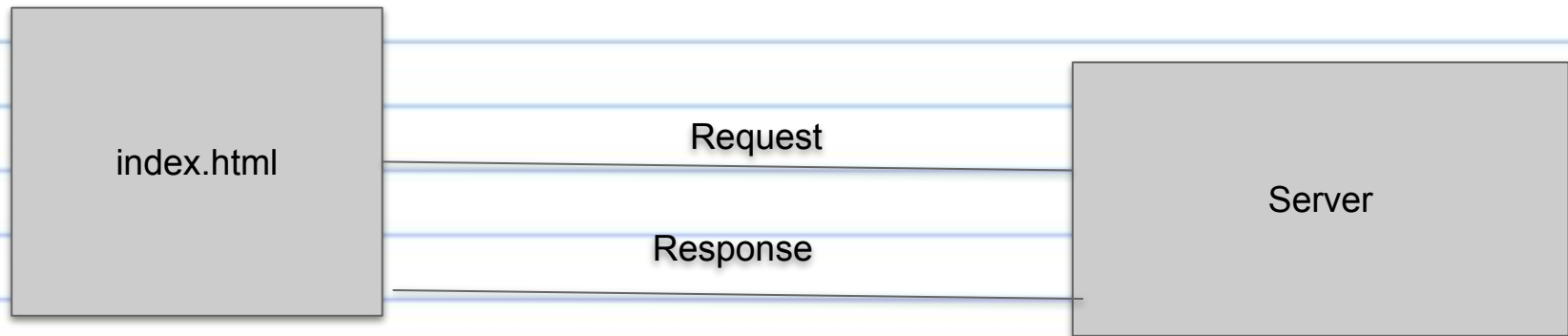
# Ohne React/JS: Multi Page Apps

`/index`

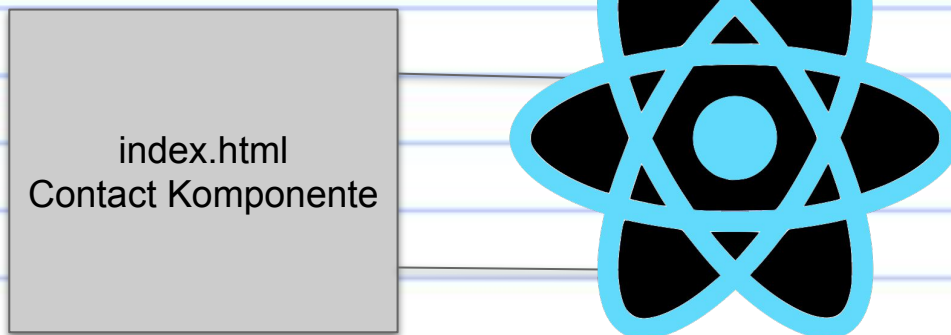


# Mit React: Single Page Apps

`/index`



`/contact`



# VS Code & Browser Demo



# Den npm Entwicklungsserver starten

```
> npm start
```

```
Compiled successfully!
```

```
You can now view my-app in the browser.
```

Local: <http://localhost:3000>

On Your Network: <http://192.168.1.22:3000>

Note that the development build is not optimized.

To create a production build, use `npm run build`.

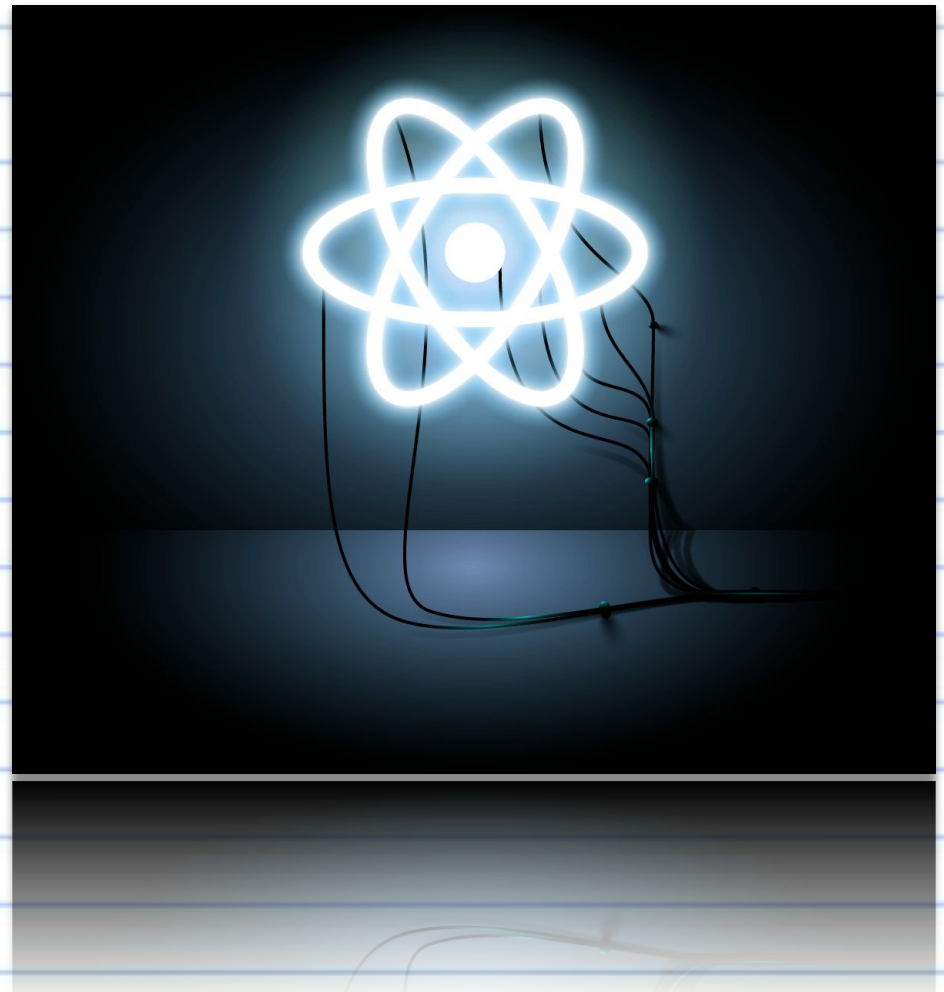
```
> Strg + C zum Stoppen des Servers
```



# Automatisch generierte Dateien aufräumen

my-app

- |— README.md
- |— node\_modules
- |— package.json
- |— .gitignore
- |— public
  - |— favicon.ico
  - |— **index.html**
  - |— manifest.json
- |— src
  - |— App.css
  - |— **App.js**
  - |— App.test.js
  - |— index.css
  - |— **index.js**
  - |— logo.svg
  - |— serviceWorker.js
  - |— setupTests.js



# VS Code & Browser Demo





# React: Nesting Components

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

# <App>: Die Mutter aller Components

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

```
class App extends Component {
```

```
  render() {
```

```
    return (
```

```
      <div className="App">
```

```
        <h1>My first React app</h1>
```

```
        <p>Welcome :)</p>
```

```
      </div>
```

```
    );
```

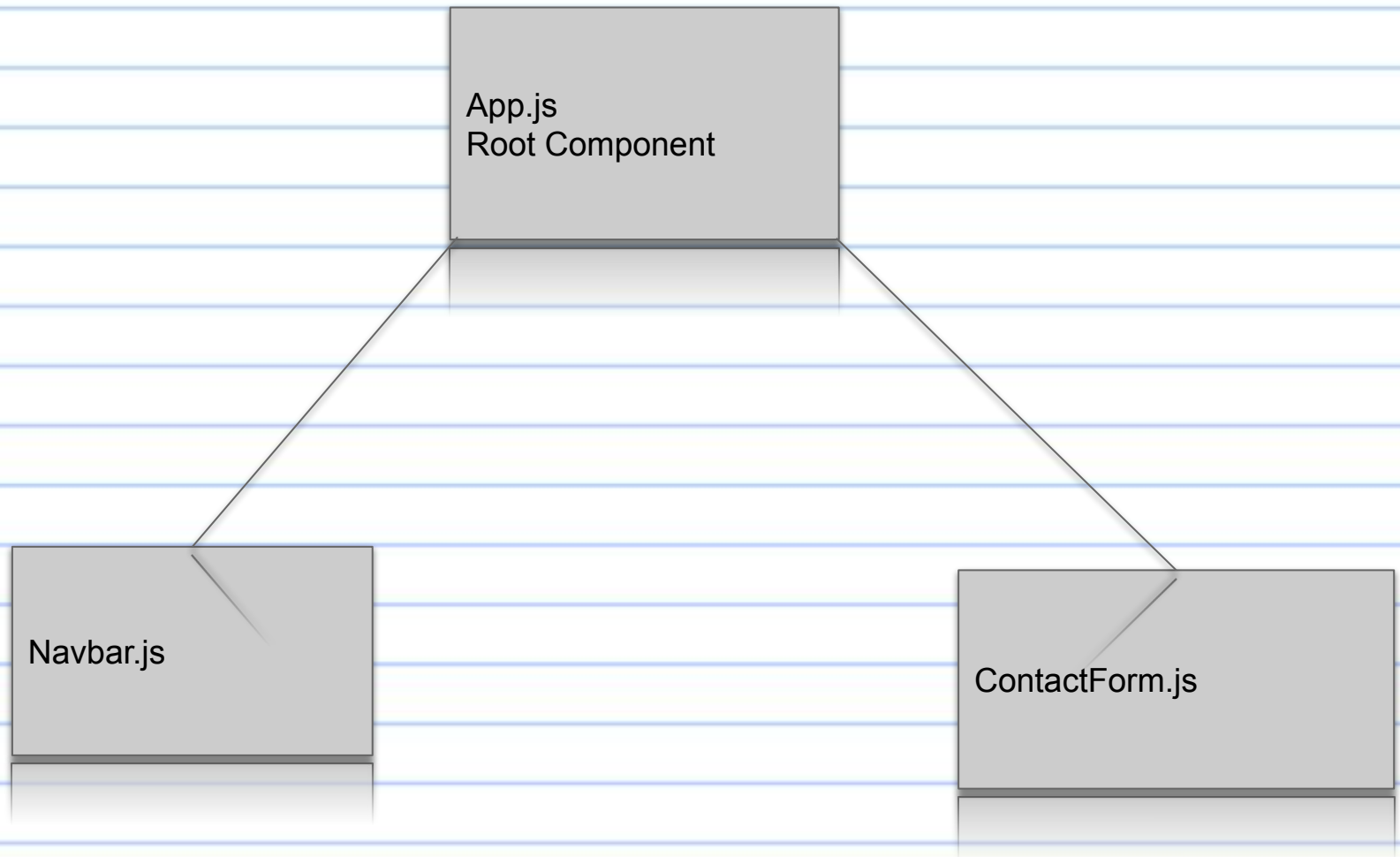
```
  }
```

```
}
```

```
export default App;
```



# React: Nesting Components



# VS Code & Browser Demo



# Neue Komponente: <Ninjas>.js

```
import React, { Component } from 'react'
```

```
class Ninjas extends Component{
```

```
  render() {
```

```
    return (
```

```
      <div className="ninja">
```

```
        <div>Name: Ryu</div>
```

```
        <div>Age: 30</div>
```

```
        <div>Belt: Black</div>
```

```
      </div>
```

```
    )
```

```
  }
```

```
}
```

```
export default Ninjas
```



# Importieren in <App>.js

```
import React, { Component } from 'react';  
import Ninjas from './Ninjas'
```

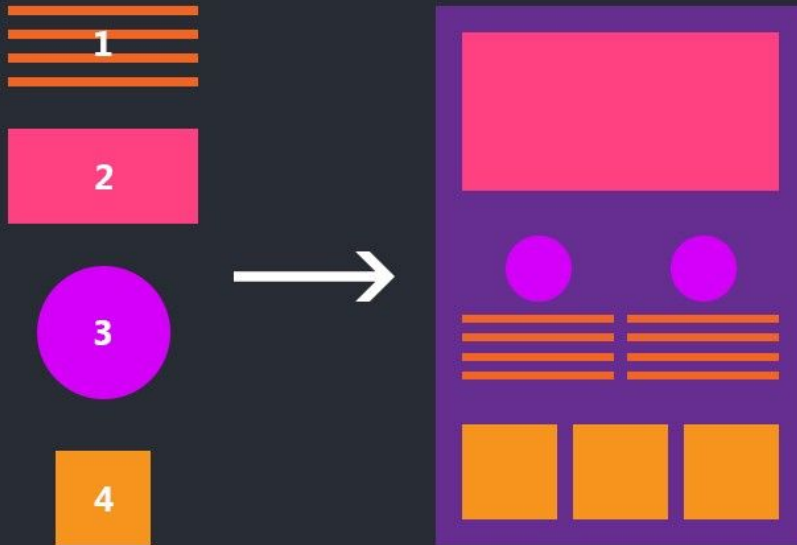
```
class App extends Component {  
  render() {  
    return (  
      <div className="App">  
        <h1>My first React app</h1>  
        <Ninjas />  
      </div>  
    );  
  }  
}
```

```
export default App;
```



# Warum nochmal verschiedene Komponenten?

## React Components



TD TECH DIAGONAL

JD TECH DIVCONVT

- Wiederbenutzbarkeit
- Kapselung
- Dry Principle: Don't repeat yourself!
- Eigene States, eigene Methoden
- Dadurch möglich, es auch innerhalb anderer Komponenten hinzuzufügen

# React: Properties

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*



# Was sind React Properties?

Ähnlich wie mit React State können wir  
Daten auch Mit Properties (kurz Props)  
Zwischen den Komponenten übergeben

Z.B. von einer Eltern-Komponente  
In eine Kind-Komponente



In unserem Beispiel von der `<App>` Komponente in die  
`<Ninjas>` Komponente. Statt hardcodierter Daten also  
besser Props!

```
<Ninjas name="Ryu" age="25 belt="black" />
```

=> erzeugt ein Objekt `{}` mit allen Übergabe Properties

# React Props in Action

```
import React, { Component } from 'react'
```

```
class Ninjas extends Component {
```

```
  render() {
```

```
    console.log(this.props);
```

```
    //const { name, age, belt } = this.props;
```

```
    return (
```

```
      <div className="ninja">
```

```
        <div>Name: { this.props.name }</div>
```

```
        <div>Age: { this.props.age }</div>
```

```
        <div>Belt: { this.props.belt }</div>
```

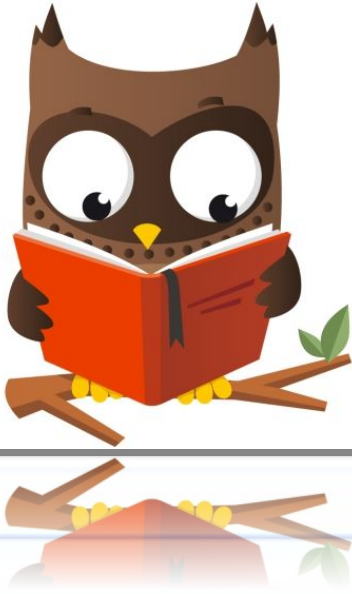
```
      </div>
```

```
    )
```

```
  }
```

```
}
```

```
export default Ninjas
```



# VS Code & Browser Demo



# JS Exkurs: "Destructurierung"

## Destructurierende Zuweisung =

Die destructurierende Zuweisung ermöglicht es, Daten aus Arrays oder Objekten zu extrahieren, und zwar mit Hilfe einer Syntax, die der Konstruktion von Array- und Objekt-Literalen nachempfunden ist.

```
let a, b, rest;
```

```
[a, b] = [10, 20];
```

```
console.log(a);
```

```
// expected output: 10
```

```
console.log(b);
```

```
// expected output: 20
```

```
[a, b, ...rest] = [10, 20, 30, 40, 50];
```

```
console.log(rest);
```

```
// expected output: Array [30,40,50]
```

Mehr Infos unter

[https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Operators/Destructurierende\\_Zuweisung](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Operators/Destructurierende_Zuweisung)



# “Destrukturierung” von React Props

```
import React, { Component } from 'react'
```

```
class Ninjas extends Component{
```

```
  render() {
```

```
    console.log(this.props);
```

```
    const { name, age, belt } = this.props;
```

```
    return (
```

```
      <div className="ninja">
```

```
        <div>Name: { name }</div>
```

```
        <div>Age: { age }</div>
```

```
        <div>Belt: { belt }</div>
```

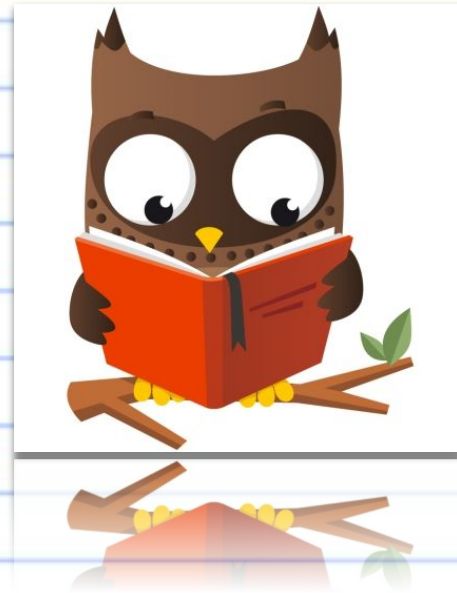
```
      </div>
```

```
    )
```

```
  }
```

```
}
```

```
export default Ninjas
```



# Vorteil von React Properties?

Wir können jetzt durch Props  
Verschiedene Daten an eine Komponente  
Liefern und haben dadurch dynamische  
Statt statische Komponenten.

```
<Ninjas name="Yoshi" age="20" belt="green" />
```

```
<Ninjas name="Ryu" age="30" belt="black" />
```

# React: Loop Property Arrays

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

# Ein Array von Props übergeben

Angenommen wir wollen nicht einzelne Ninja Daten übergeben, sondern lieber Gleich eine ganze Liste mit einem Array:

Dazu können wir im state ein Objekt mit Einer Property "ninjas" anlegen, dass Dieses Array enthält. Jeder Teil des Arrays ist wieder ein Objekt.

Statt mehrere Komponenten aufzurufen brauchen wir sie nur einmal einbinden und dieses ninja-Array mit `{this.state.ninjas}` zu übergeben.

Die weitere Logik befindet sich dann in der Ninja Komponente.





# Ein Array von Props übergeben

```
class App extends Component {  
  state = {  
    ninjas: [  
      { name: 'Ryu', age: 30, belt: 'black', id: 1 },  
      { name: 'Yoshi', age: 20, belt: 'green', id: 2 },  
      { name: 'Crystal', age: 25, belt: 'pink', id: 3 }  
    ]  
  }  
  
  render() {  
    return (  
      <div className="App">  
        <h1>My first React app</h1>  
        <Ninjas ninjas={this.state.ninjas}/>  
      </div>  
    );  
  }  
}
```



# VS Code & Browser Demo



# JS Exkurs: `Array.prototype.map()`

Die `map()` (engl. *abbilden*) Methode wendet auf jedes Element des Arrays die bereitgestellte Funktion an und gibt das Ergebnis in einem neuen Array zurück.

```
const array1 = [1, 4, 9, 16];
```

```
// pass a function to map
```

```
const map1 = array1.map(x => x *  
2);
```

```
console.log(map1);
```

```
// expected output: Array [2, 8,  
18, 32]
```

Mehr Infos unter

[https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global\\_Objects/Array/map](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array/map)



# Ein Array von Props auf die Ninja Komponente "mappen" => Schleife!

Wir benutzen Destructurierung um unsere Property auszupacken.

Danach benutzen wir die Map-Funktion, Um über das Array zu loopen und die Jeweilige passende Information in einem JSX zu exportieren, wo wir auf die einzelnen State-Properties zugreifen können

Im Return fügen wir dann nur noch das "gemappte" Element ein und haben automatisch für jedes Array-Item eine JSX-Komponente gerendert!

Jede Komponente ist individuell, damit React weiß, was sich geändert hat in der Zukunft!



# Ein Array von Props auf die Ninja Komponente "mappen" => Schleife!

```
render() {  
  const { ninjas } = this.props;  
  const ninjaList = ninjas.map(ninja => {  
    return (  
      <div className="ninja" key={ninja.id}>  
        <div>Name: { ninja.name }</div>  
        <div>Age: { ninja.age }</div>  
        <div>Belt: { ninja.belt }</div>  
      </div>  
    )  
  });  
  return (  
    <div className="ninja-list">  
      { ninjaList }  
    </div>  
  )  
}
```



# React: Komponenten ohne State

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*

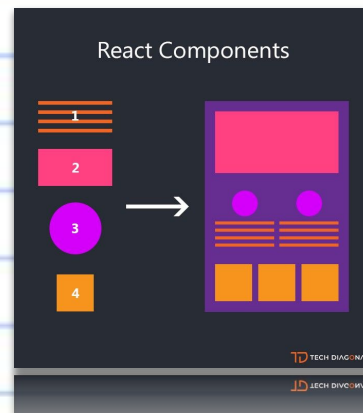
# Container vs. UI Komponenten

## Container Komponenten

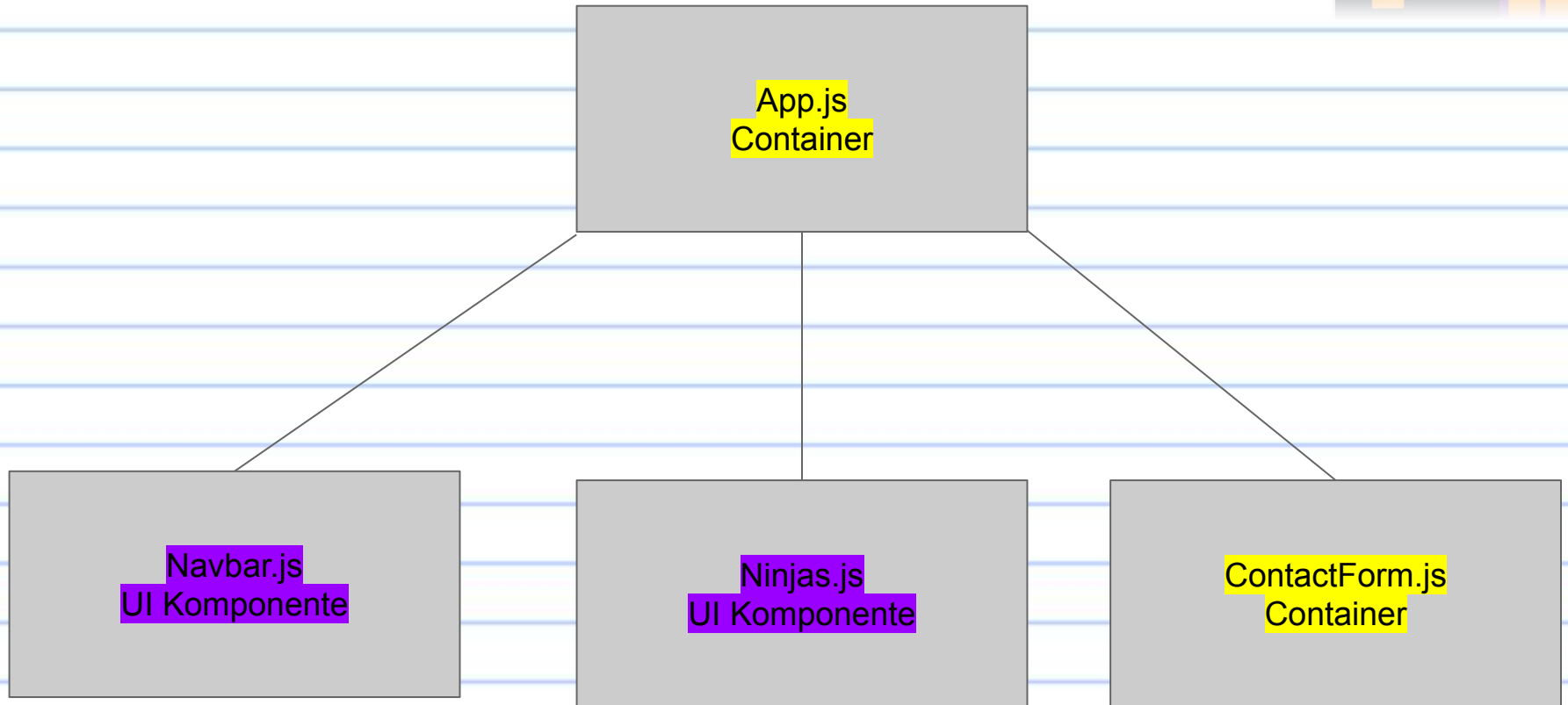
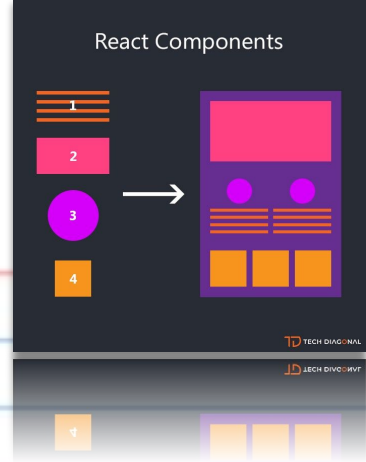
- Enthalten State
- Enthalten Lifecycle Hooks
- Sind nicht am UI beteiligt
- Werden als Klasse erzeugt
- A.k.a. "Classbase" Components

## UI Komponenten

- Enthalten keinen State
- Enthalten Daten als Property (wahrscheinlich von einem Container State)
- Sind nur am UI beteiligt
- Werden als Funktion erzeugt
- A.k.a. "Stateless" Components



# Container vs UI Komponenten



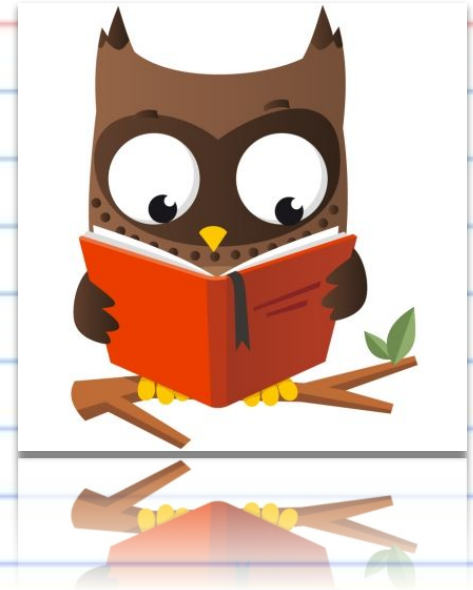


# Refactoring der Ninjas Komponente

Da diese Komponente keinen State braucht  
(bekommt State-Props über die <App>)

Brauchen wir keine Klasse sondern können  
Sie zu einer Arrow-Funktion umbauen:

```
import React from 'react';  
  
const Ninjas = (props) => {  
  const { ninjas } = props;  
  // alles andere bleibt gleich...  
}
```



# Destrukturierung in Funktionen

Auch hier können wir wieder Destrukturierung nutzen, um unseren Code zu vereinfachen:

```
const Ninjas = ({ninjas}) => {  
const { ninjas } = props;  
  // alles andere bleibt gleich...  
}
```

A yellow square with the letters 'JS' in a large, bold, black sans-serif font. Below the square is a faint, semi-transparent reflection of the square and the letters 'JS'.

# VS Code & Browser Demo

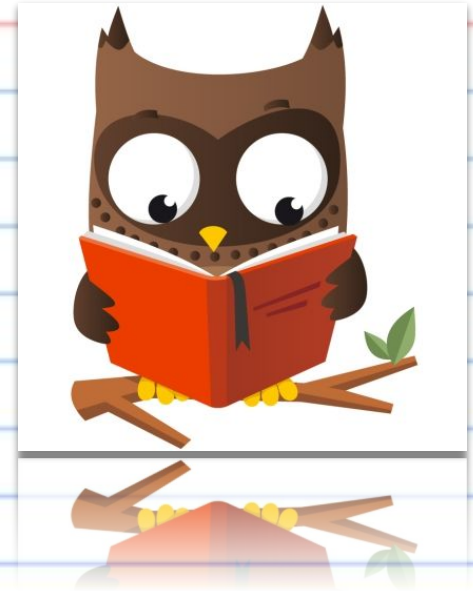


# React: Conditional Output

*Martina Freundorfer  
Hochschule für Technik &  
Wirtschaft (HTW) Berlin*


# If-Statements in React

```
const ninjaList = ninjas.map(ninja => {  
  if (ninja.age > 20) {  
    return (  
      <div className="ninja" key={ninja.id}>  
        <div>Name: { ninja.name }</div>  
        <div>Age: { ninja.age }</div>  
        <div>Belt: { ninja.belt }</div>  
      </div>  
    )  
  } else {  
    return null;  
  }  
});
```



# JS: Bedingter (ternärer) Operator

Der **bedingte (ternäre) Operator** ist der einzige Operator in JavaScript, der drei Operanden hat. Er wird häufig als Kurzform eines `if` Statements genutzt.



```
function getFee(isMember) {  
    return (isMember ? '$2.00' : '$10.00');  
}  
  
console.log(getFee(true));  
// expected output: "$2.00"  
  
console.log(getFee(false));  
// expected output: "$10.00"  
  
console.log(getFee(1));  
// expected output: "$2.00"
```

Mehr Infos unter

[https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Operators/Conditional\\_Operator](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Operators/Conditional_Operator)

# Refactoring der Ninjas Komponente?

```
return (  
  <div className="ninja-list">  
    {  
      ninjas.map(ninja => {  
        return ninja.age > 20 ? (  
          <div className="ninja" key={ninja.id}>  
            <div>Name: { ninja.name }</div>  
            <div>Age: { ninja.age }</div>  
            <div>Belt: { ninja.belt }</div>  
          </div>  
        ) : null  
      })  
    }  
  </div>  
) ;
```



# Refactoring der Ninjas Komponente?

Hier können Sie selbst entscheiden,  
Welche Variante Ihnen besser gefällt!

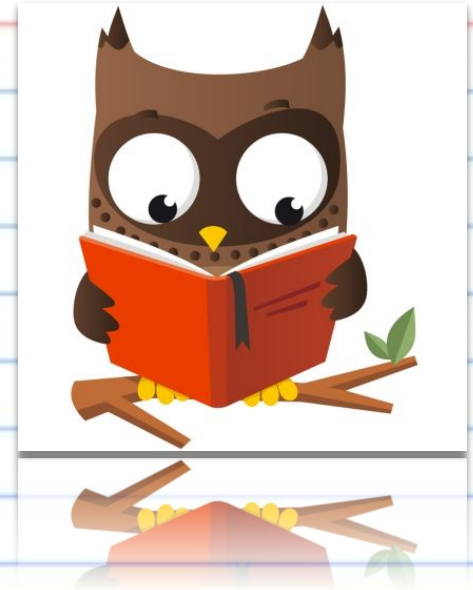
Die normale JavaScript Schreibweise

```
if (condition) {...} else {...}
```

oder der kürzere bedingte (ternäre) Operator

```
(condition) ? true : false
```

Wir werden beides in der Zukunft sehen & benutzen!





# VS Code & Browser Demo



# Credits & Mehr Lesestoff



Die Kursdateien für das React & Redux Tutorial

<https://github.com/htw-web/react-redux-complete-playlist>

Shaun "The Net Ninja" Pelling (Github: iamshaunjp)

<https://www.youtube.com/channel/UCW5YeuERMmlnqo4oq8vwUpq>

<https://github.com/iamshaunjp>

Getting Started with React - An Overview and Walkthrough  
Tutorial (including create-react-app)

<https://www.taniarascia.com/getting-started-with-react/>

Vielen Dank für Ihre Aufmerksamkeit!