React

Grundlagen für Entwickler

trinnovative.

Software, Cloud & KI einfach gemacht.

17.10.2023

DAS BIN ICH - Michael Schmidt

Fullstack Entwickler mit Fokus auf Frontend

- Seit 20 Jahren in Regensburg und seit 5 Jahren bei der trinnovative GmbH
- Bachelor in Informatik an der OTH Regensburg abgeschlossen
- Master in Medieninformatik an der Uni Regensburg (pausiert)
- Unterstützte aktuell die Firma Horsch im Bereich Frontendentwicklung mit React

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Universität Regensburg

Software, Cloud & KI einfach gemacht.

trinnovative.

React - „The library for web and native UI“

Vorstellung trinnovative GmbH

0 - JavaScript Basics für React

1 - Einführung

2 - State

3 - JSX

4 - Component und Hooks

5 - Entwicklungs Tools & Projekt Setup

React - „The library for web and native UI“

6 - Hands on: Todo App

7 - Packages & React Ecosystem

8 - API Abfragen

9 - Side Effects

10 - Context

11 - Ausblick

12 - Hands on: State of the Art Todo App

VORSTELLUNG TRINNOVATIVE

INNOVATIVER DIENSTLEISTER FÜR SOFTWARE, CLOUD & KI

- Alle Branchen, technologieunabhängig
- Schnell wachsendes Team von aktuell 16 Spezialisten
- Zahlreiche Abschlussarbeiten, Vorträge und Veröffentlichungen

DGO
Digitale Gründerinitiative
OBERPFALZ

Cluster
Mobility & Logistics

AIR
ARTIFICIAL INTELLIGENCE
REGENSBURG

Software, Cloud & KI einfach gemacht.

trinnovative.

EINSATZ VON REACT

Warum?

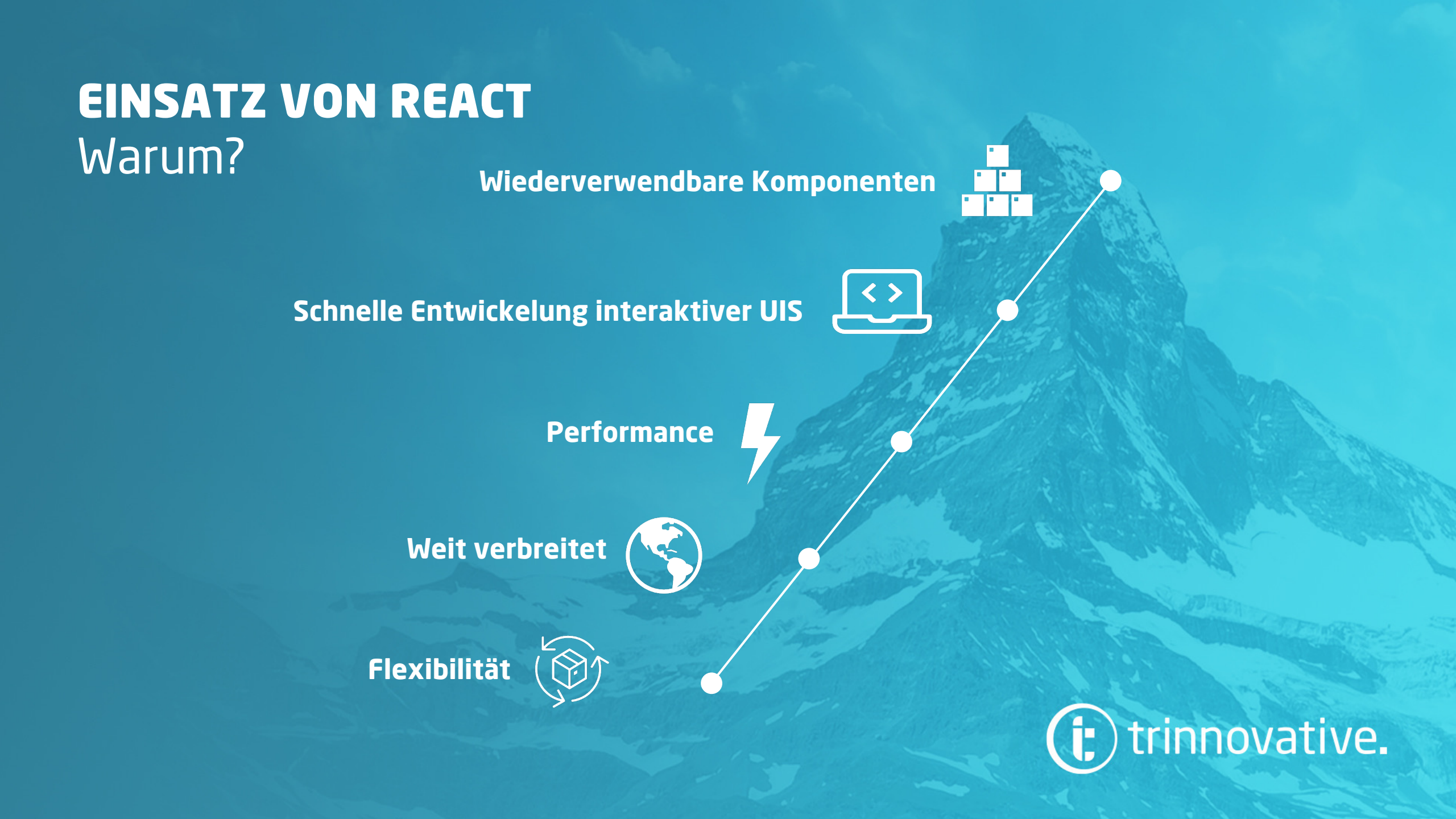
Wiederverwendbare Komponenten

Schnelle Entwickelung interaktiver UIS

Performance

Weit verbreitet

Flexibilität



trinnovative.

0 - JavaScript Basics für React - JS

- Kerntechnologie (mit HTML & CSS) fürs Web
- High-Level
- Client & Server seitig (Browser bzw. Node.js)
- Single Threaded & Functional
- Basics ähnlich zu anderen Sprachen (If, Schleifen, Arrays, etc.)

JS

trinnovative.

0 - JavaScript Basics für React - TypeScript

- Free & Open Source von Microsoft entwickelt
- Statische Typisierung von JavaScript zur **Kompilierzeit**
- TS Code -> wird zu JS kompiliert wird
- Persönliche Meinung:
  - JavaScript = schweres TypeScript
  - TypeScript = einfaches JavaScript

trinnovative.

0 - JavaScript Basics für React - Destructuring

- Syntax fürs “entpacken” / anlegen von:
  - Properties aus Objekten

```
// Ohne Destructuring:
const protocol = window.location.protocol;
const host = window.location.host;
const pathname = window.location.pathname;

// Mit Destructuring:
const { protocol, host, pathname } = window.location;
```

trinnovative.

0 - JavaScript Basics für React - Destructuring

- Syntax fürs “entpacken” / anlegen von:
  - Values aus Arrays

```
// Für Arrays
const [a, b, c] = [1, 2, 3];
console.log("Var A: ", a); // "Var A: 1”

// Return values
const [result, logs] = someLongComputation();
```

trinnovative.

0 - JavaScript Basics für React – Arrow Functions

- Syntax für anonyme (namenlose) Funktionen
- Used to invoke function immediately after initialization

```
const multiply = (a, b) => {
  return a * b;
};

// Kurzform
const multiply = (a, b) => a * b;
```

trinnovative.

0 - JavaScript Basics für React – Callbacks

- Callback Functions:
  - Function that is passed to another function as argument
  - And can be invoked during the execution of the higher order function

```
const isEven = (n) => {
  return n % 2 == 0;
}

const printMsg = (evenFunc, num) => {
  const isNumEven = evenFunc(num);
  console.log(`The number ${num} is an even number: ${isNumEven}.`)
}

// isEven übergeben als callback function
printMsg(isEven, 4);
```

trinnovative.

0 - JavaScript Basics für React – Higher Order Functions

- .map()
- .forEach() (
- .reduce()
- .filter()

```
const finalMembers = ['Taylor', 'Donald', 'Don', 'Natasha', 'Bobby'];

const announcements = finalMembers.map((member) => {
  return member + ' joined the contest.';
});

// "Taylor joined contest", etc.
```

trinnovative.

0 - JavaScript Basics für React – String Templates

- String Interpolation via Backticks ` ` mit ${ }

```
const name = "React";
const message = `Hello, ${name}!`; // "Hello, React!"
```

trinnovative.

0 - JavaScript Basics für React – Spread Operator

- Syntax fürs kopieren von Values aus Arrays oder Objekten

```
// Array
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];

// Object
const myVehicle = {
  brand: 'Ford',
}

const updateMyVehicle = {
  type: 'car',
}

const myUpdatedVehicle = {...myVehicle, ...updateMyVehicle}
```

0 - JavaScript Basics für React – Import / Exports

- Wie verwendete ich Variablen / Funktionen in anderen Datein?
- (Named) Exports:

```
const foo = 1;
function bar() {
  return 2;
}

export { foo, bar };
```

```
// Oder
export const foo = 1;
export function bar() {
  return 2;
}
```

trinnovative.

0 - JavaScript Basics für React – Import / Exports

- Wie verwendete ich Variablen / Funktionen in anderen Datein?
- (Named) Imports:

```
// Imports in anderen Datei
import { foo, bar } from "./mymodule.js";

// Mit Bundler (e.g. Webpack)
import { foo, bar } from "./mymodule";
```

trinnovative.

0 - JavaScript Basics für React – Import / Exports

- Wie verwendete ich Variablen / Funktionen in anderen Datein?
- (Default) Exports:

```
// Nur 1 default export pro Module
export default function Main() {
  return <div>Main</div>;
}
```

```
// Oder
function Main() {
  return <div>Main</div>;
}

export default Main;
```

trinnovative.

0 - JavaScript Basics für React – Import / Exports

- Wie verwendete ich Variablen / Funktionen in anderen Datein?
- (Default und Named) Imports:

```
import Main, { foo, bar } from 'mymodule.js’;
```

trinnovative.

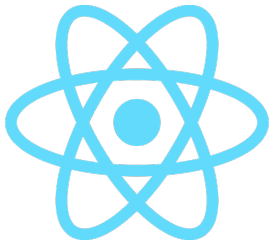
0 - JavaScript Basics für React – Recap

- Destructuring
- Arrow Functions
- Callbacks
- Higher Order Functions
- String Templates
- Spread Operator
- Import / Exports

trinnovative.

1 - Einführung - React Eigenschaften

- Free & Open Source, von Facebook 2013 entwickelt
- Betrieben (meist) Client-Side im Browser / Webserver
- Entwicklung lokal in einem Node.js Environment
- State Based
- Component Based
- Declarative

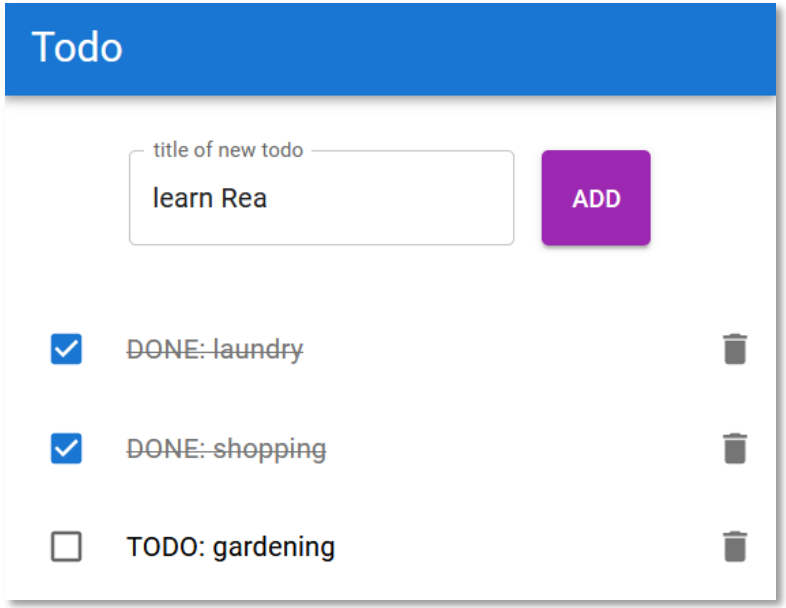


trinnovative.

1 - Einführung - State Based

- Daten Modell welches den App Zustand beschreibt
- useState()

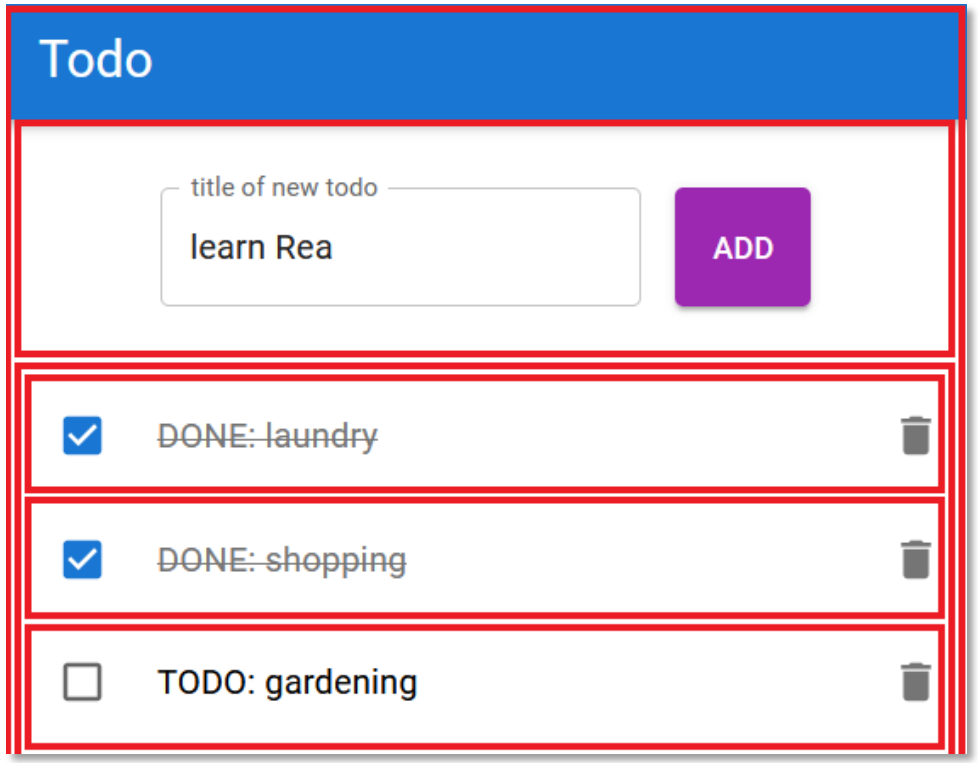
```
// Kompletter Zustand einer Todo-App
{
  "newTitleInput": "learn Rea",
  "todos": [
    { "id": 1, "title": "laundry", "completed": true },
    { "id": 2, "title": "shopping", "completed": true },
    { "id": 3, "title": "gardening", "completed": false }
  ]
}
```



trinnovative.

1 - Einführung - Component Based

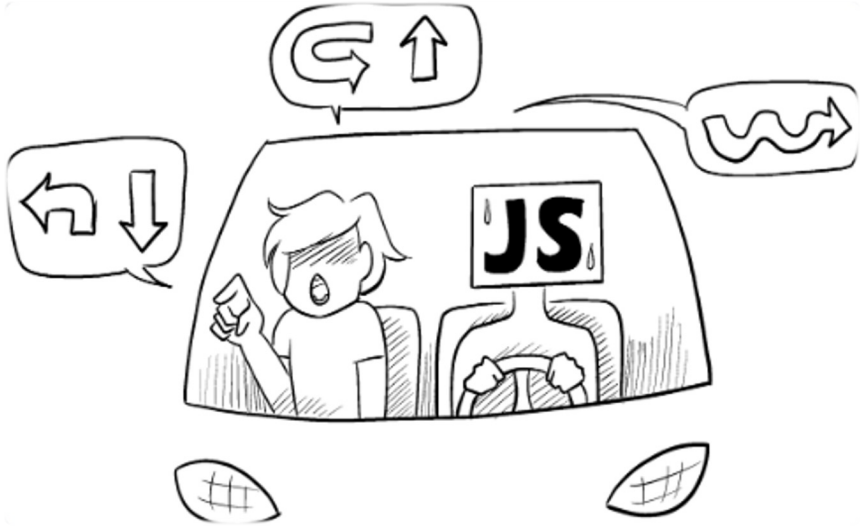
- Wiederverwendbare UI Elemente
- Kommunikation zwischen Parents und Child Elements über Properties (Props) und Events

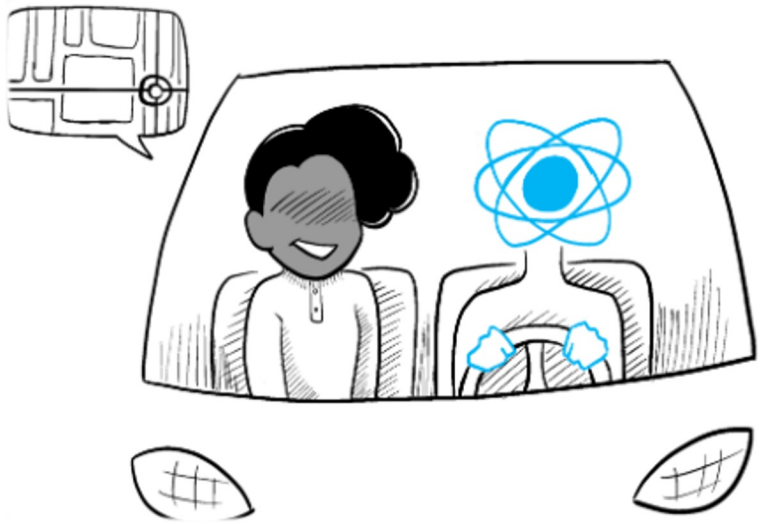


trinnovative.

1 - Einführung - Declarative

- Beschreiben der finalen UI für jeden visuellen State, anstelle von Micromangen der UI (imperative z.B. JQuery)
- https://react.dev/learn/reacting-to-input-with-state



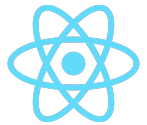


trinnovative.

1 - Einführung - Imperative vs Declarative

```
updatePrice(price) {
  $("#price-label").val(price);
  $("#price-label").toggleClass('expansive', price > 100);
  $("#price-label").toggleClass('cheap', price < 100);

  // also remember to update UI depending on price
  updateTotalPrice();
  ...
}

updateVolume(volume) {
  $("#volume-label").val(volume);
  $("#volume-label").toggleClass('high', volume > 1000);
  $("#volume-label").toggleClass('low', volume < 1000);

  // also remember to update UI depending on volume
  updateTotalPrice();
  ...
}

updateTotalPrice() {
  const totalPrice = price * volume;
  $("#total-price-label").val(totalPrice);
  ...
}
```

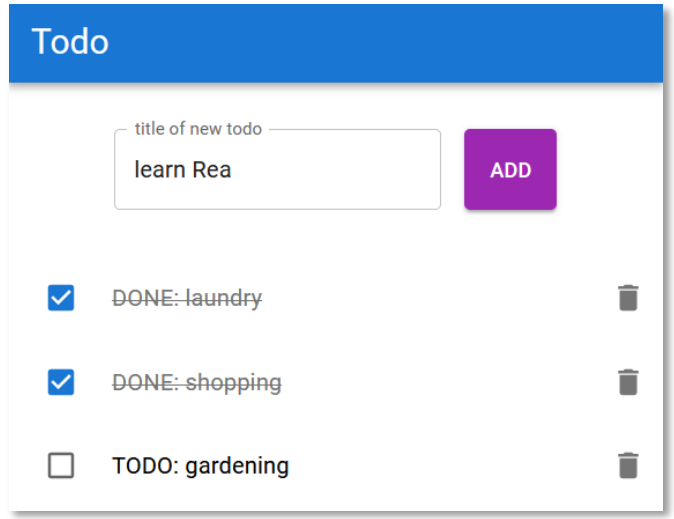

```
function MyComponent() {
  const [price, setPrice] = useState(0);
  const [volume, setVolume] = useState(0);
  const totalPrice = price * volume;

  return (
    <div>
      <Label value={price} className={price > 100 ? "expensive" : "cheap"} />
      <Label value={volume} className={volume > 1000 ? "high" : "low"} />
      <Label value={totalPrice} />
    </div>
  );
}
```



trinnovative.

1 - **Einführung** - Declarative

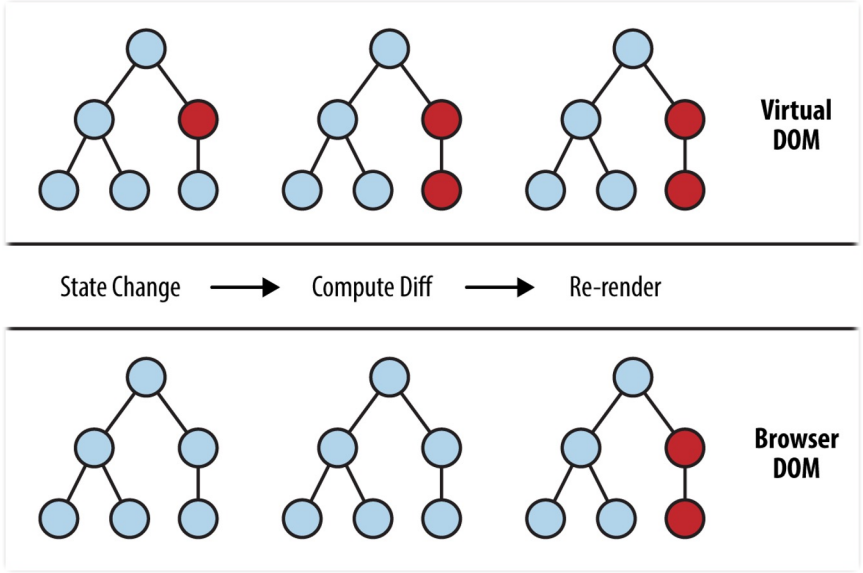
1. **Identifiy** components different visual states
2. **Determine** what triggers state changes
3. **Represent** state in memory using useState
4. **Remove** any non-essential state variables
5. **Connect** event handlers to set state



trinnovative.

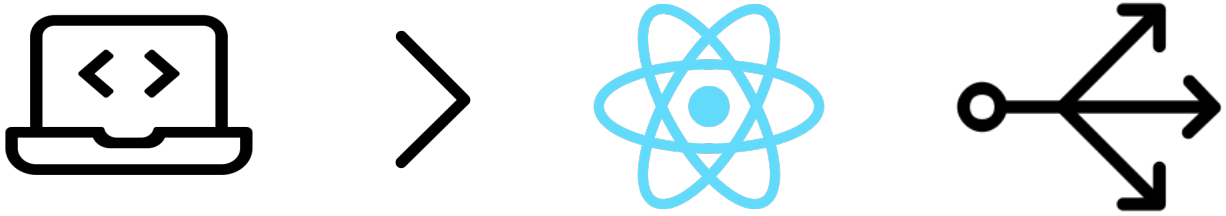
1 - Einführung - Rendering

- Virtual DOM (**D**ocument **O**bject **M**odel) & React API
- Virtual representation of a UI is kept in memory and synced with destination UI
- Enables you to tell the declarative React API what state you want
- https://legacy.reactjs.org/docs/faq-internals.html



trinnovative.

1 - Einführung - Rendering



React Codebase

Virtual DOM Layer

React Native

React PDF

[...]

remotion

(mp4 Rendering)

chrome

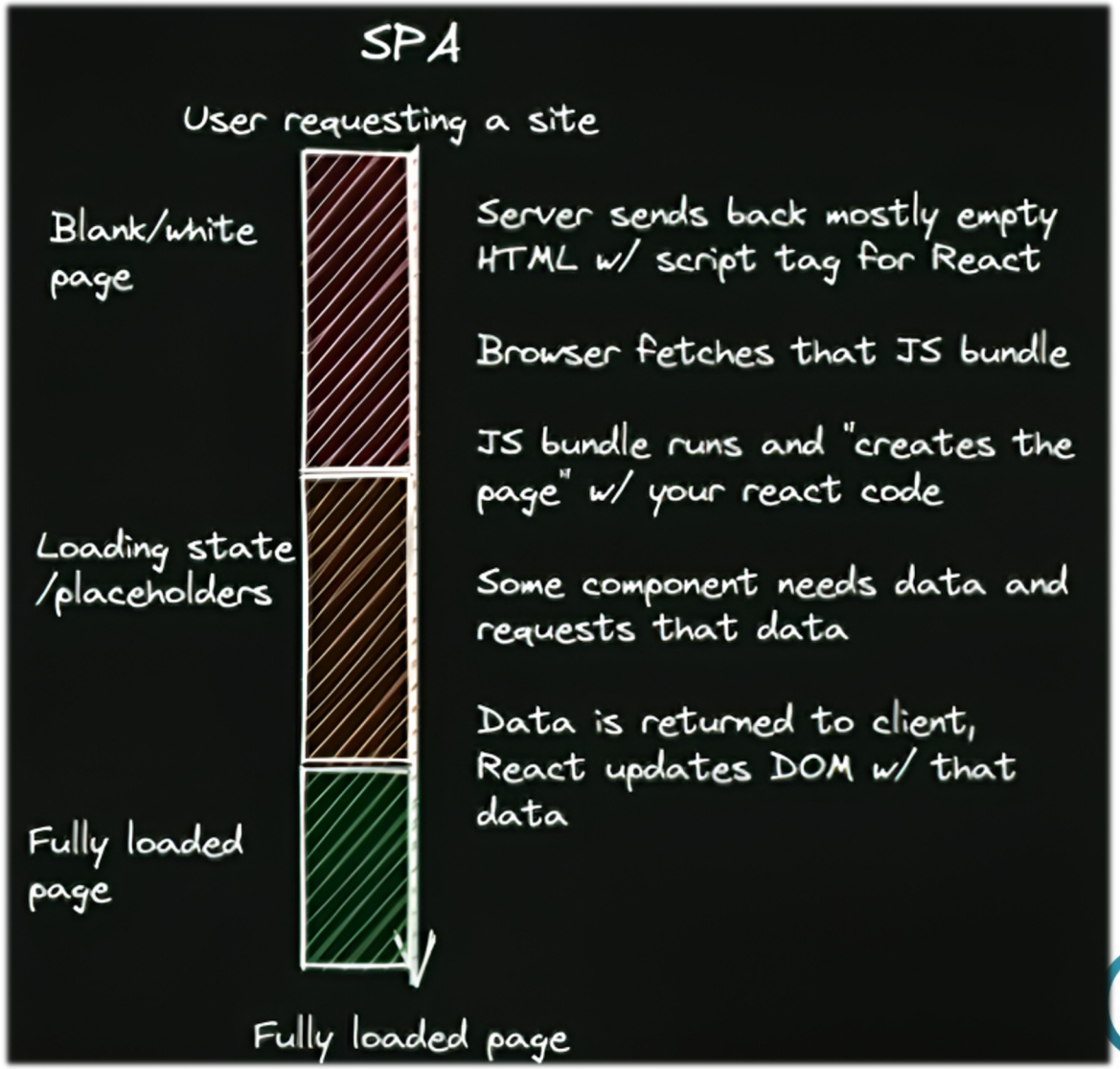
(Browser)

slack

(Desktop Apps)

trinnovative.

1 - Einführung - Rendering (im Browser)



Theo Brown - https://www.youtube.com/watch?v=Ao8F3FypsbI&ab_channel=Theo-t3%E2%80%A4gg

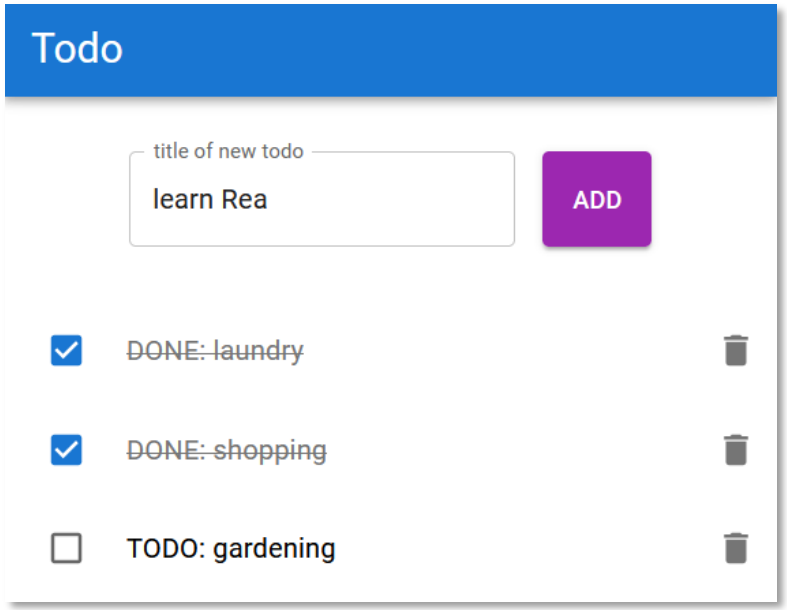
trinnovative.

1 - Einführung - Recap

- State Based
- Component Based
- Declarative
- Rendering

2 - State - Basics

- Einsatz wenn Components Daten zwischen Renders brauchen
- Variable vs State
- State ist private in einer Component
- https://react.dev/learn/state-a-components-memory



trinnovative.

2 - State - State Hook

- Parameter: Intialer State
- Return Values (Destructured): Aktueller State, Setter Funktion

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

function CounterDisplay() {
  const [counter, setCounter] = useState(0);

  return ...
};
```

trinnovative.

2 - State - Immutable State

- State sollte als immutable (unveränderbar) angesehen werden
- Wenn wir z.B. Objects / Arrays als state haben ..
  - .. Könnten wir sie via Call by reference mutaten ❌
- React vergleicht aber immer alte / neue State Reference
  - Falls Reference sich nicht ändert -> kein Rerendering
- Demo

trinnovative.

2 - State - Input State (Controlled)

- value={text} binds state to input
- onChange: HTML Event
- setText(...) updates state when input changes

```
function App() {
  const [text, setText] = useState('');

return (
    <input
      value={text}
      onChange={(event) => setText(event.target.value)}
    />
  );
}
```

2 - State - Input State (Uncontrolled)

- Optionen zum managen von Input State
  - Form Submit (via FormData)
  - External Libraries (z.B. react-hook-form, formik)

2 - State - Recap

- Basics
- useState Hook
- Immutable State
- Input State

3 - JSX - Basics

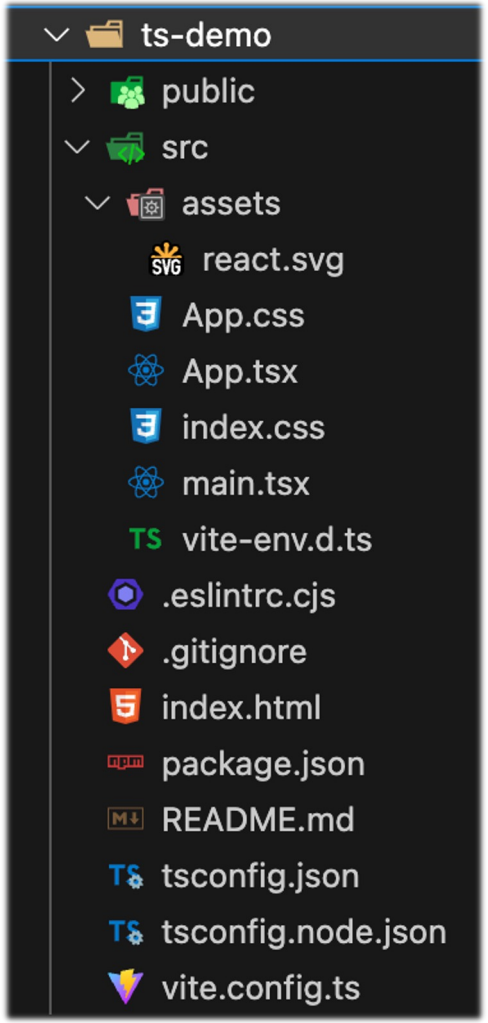
- JSX = **J**ava**S**cript **X**ML
- Dateiformat: MyButton.jsx (bzw. in TS .tsx)
- Valid Elements in JSX
  - String, Number, Arrays of other Elements
  - HTML Elements (, <img>, ...)
  - Other JSX Components
  - null, undefined, true, false (these are not rendered)

```
// Neither a string nor HTML -> JSX
const element = <h1>Hello, world!</h1>;
```

trinnovative.

3 - JSX - React Projekt Aufbau

- Bitte TS in Projekten benutzen
- Demo package.json
- Demo Aufbau App.jsx



vite.js (TS) Starter Project

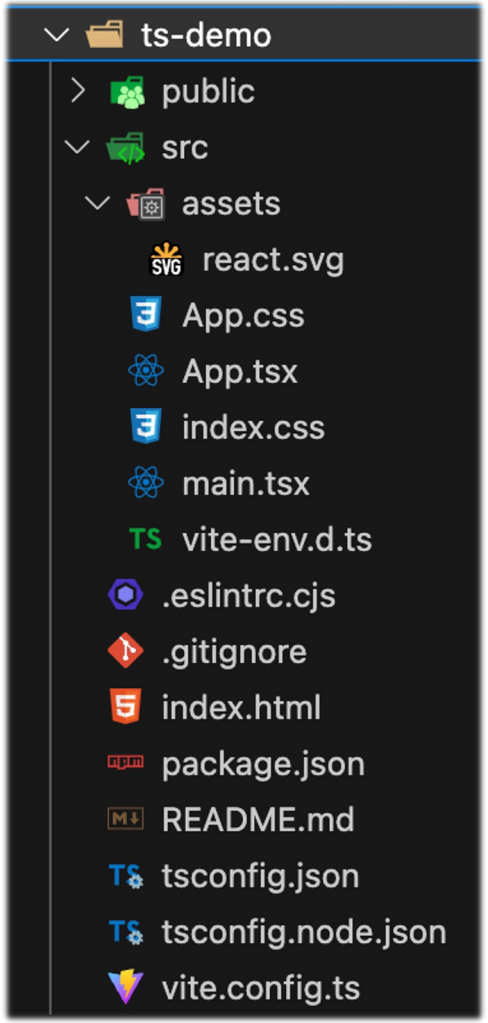
trinnovative.

3 - JSX - Property Names

- Einige Element Properties haben andere Namen als in HTML
  - className (instead of class)
  - htmlFor (instead of for)
  - Auch fürs styling: marginBottom (instead of margin-bottom)
  - …

3 - JSX - Styling

- Möglichkeiten:
  - CSS Datei
  - Style Definitionen am Anfang der Datei
  - Inline style im Component Template
  - Third Party styling / Component Libraries



vite.js (TS) Starter Project

trinnovative.

3 - JSX - Styling

- Mit CSS

```
// in TodoItem.jsx
import './TodoItem.css';

...
<li
  className={
    isCompleted
      ? 'TodoItem TodoItem--Completed'
      : 'TodoItem'
  }
>
  <span className="TodoItem__Title">...</span>
</li>
```

3 - JSX - If / Else

- Conditional Rendering

```
let statusMessage;

if (gameActive) {
  statusMessage = <div>score: {score}</div>;
} else {
  statusMessage = (
    <div>Game over. Final score: {score}</div>
  );
}

return (
  <div>
    {statusMessage}
  </div>
);
```

3 - JSX - If / Else (Ternary)

- Ternary Operator in JSX
  - condition ? true : false

```
<div>
  <h1>Foo Game</h1>
  {gameActive ? (
    <div>score: {score}</div>
  ) : (
    <div>Game over. Final score: {score}</div>
  )}
</div>
```

trinnovative.

3 - JSX - If / Else (Ternary)

- If ohne else

```
<div>{isError ? <div>{errorMessage}</div> : null}</div>
// Äquivalent
<div>{isError && <div>{errorMessage}</div>}</div>
```

trinnovative.

3 - JSX - Listen

- Mehrere Elemente können via Array / Loop gerendert werden

```
const elements = [
  <li>foo</li>,
  <li>bar</li>,
  <li>baz</li>,
];
...
<ul>
  {elements}
</ul>
```

```
const initialTodos = [
  { id: 1, title: 'groceries', completed: false },
  { id: 2, title: 'cooking', completed: true },
  { id: 3, title: 'gardening', completed: false },
];

function TodoApp() {
  const [todos, setTodos] = useState(initialTodos);

  return (
    <ul>
      {todos.map((todo) => (
        <li key={todo.id}>{todo.title}</li>
      ))}
    </ul>
  );
}
```

3 - JSX - Binding Content / Properties

- Basic Types (numbers, strings)

```
const dateString = new Date().toLocaleDateString();
<div>curent date: {dateString}</div>
```

- Change XML to JS for property

```
<a href={'https://en.wikipedia.org/wiki/' + articleName}>
wikipedia article
</a>

<button disabled>Disabled Button</button>
```

trinnovative.

3 - JSX - Binding Events

- Liste von Browser Events

```
function sayHello() {
  alert('hello world');
}

<button onClick={() => sayHello()}>Say Hello</button>
```

trinnovative.

3 - JSX - Binding Events

- Accessing event object (z.B. MouseEvent)

```
<button
  onClick={(event) => {
    console.log(event);
}}
>
  Click me
</button>
```

trinnovative.

3 - JSX - Binding Events

- Event Handler muss eine **Function** sein (kein function call)

```
// Ok
<button onClick={(event) => handleEvent(event)}>
click
</button>

// Ok
<button onClick={handleEvent}>click</button>

// Nicht Ok
<button onClick={handleEvent()}>click</button>
```

trinnovative.

3 - JSX - Binding Events (Prevent Default)

- Default behaviour of **submit in form**: Send data to server 🤔

```
<form
  onSubmit={(event) => {
    event.preventDefault();
    // handle submit with custom logic
  }}
>
  ...
  <button type="submit">submit</button>
</form>
```

trinnovative.

3 - JSX - Recap

- Basics
- Component Aufbau
- Binding Components / Properties
- Property Names
- Fragments / Tags
- Styling
- If / Else (Ternary)
- Listen
- Binding Events

trinnovative.

4 - Components und Hooks - Basics

- Components: UI / View
- Hooks: Handhaben Logik in der View
  - Funktion die in einer Component aufgerufen werden kann
  - z.B. useState, useEffect, weitere
  - Regeln: https://legacy.reactjs.org/docs/hooks-rules.html

4 - Components und Hooks - Custom Components

- Warum? **Reusability und Code Structuring**
- Beispiele Components:
  - Button, Navbar, TodoList, TodoItem, ...

```
return (
  <Container maxWidth="xl">
    <SearchBar
      searchText={searchText}
      handleInput={handleInput}
      handleClear={handleClear}
    />
    <DataTable
      employees={employees}
      actualCalendarDate={actualCalendarDate}
      absences={absences}
    />
  </Container>
);
```

nnovative.

4 - Components und Hooks - Custom Components

- Demo (wichtigste!)

vite.js (TS) Starter Project

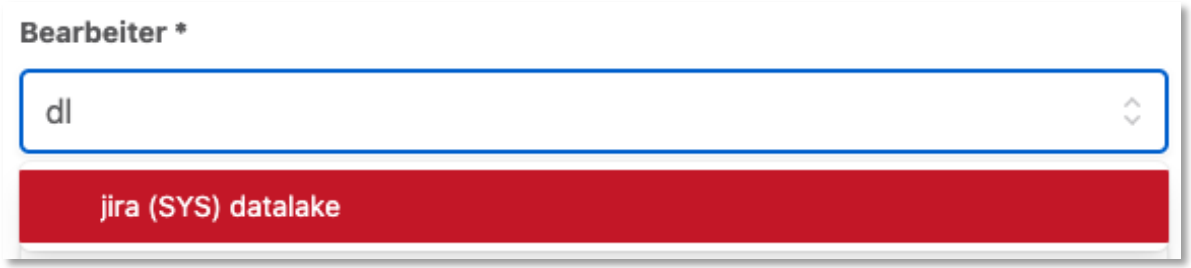
trinnovative.

4 - Components und Hooks - Custom Hooks

- Teilen Logik zwischen Components
- Weitere Eigenschaften:

https://react.dev/learn/reusing-logic-with-custom-hooks#extracting-your-own-custom-hook-from-a-component

- useDebounce
  - Oft bei Async Input Felder die mit API verbunden sind (Input Buffering)



trinnovative.

4 - Components und Hooks - Third Party

- Components und Hooks aus Libraries:
  - MUI (Material-UI): <Button>
  - React Router: <Route>, <Link>
  - React Query: useQuery
  - React-Hook-Form: useForm
  - ...

trinnovative.

4 - Components und Hooks - Recap

- Basics
- Custom
- Third Party

trinnovative.

5 - Entwicklungstools & Projekt Setup - VS Code

- Open Source IDE (von Microsoft)
- Entkoppelt von Visual Studio
- https://code.visualstudio.com/

Visual Studio Code

trinnovative.

5 - Entwicklungstools & Projekt Setup - VS Code Extensions

- Must haves:
  - Prettier (code formatter)
  - ESLint (linter)
  - Pretty TypeScript Errors
  - (Extension für Libraries die im Projekt verwendet wird)

- Persönliche Empfehlungen:
  - vscode-icons
  - Auto Rename Tag
  - Color Highlight
  - ES7+ React/Redux/React-Native snippets
  - Turbo Console Log

VS Code Sidebar

trinnovative.

5 - Entwicklungstools & Projekt Setup - React Setup

- https://react.dev/learn/start-a-new-react-project
- React empfiehlt offziell direkt Next.js
- Grobe Empfehlung:
  - Frei zugängliche Web App im Netz: Next.js
    - SEO
    - Serverside Rendering
    - Biased Features inklusive (Routing, Image loading, etc.)
  - Kleine/interne React Apps mit Basic Features: Vite.js

trinnovative.

6 - Hands on - Todo App

- Das “Hello World” der React Welt 🌍
- Setup mit VS Code, Vite.js & (TypeScript via Vite.js Template)
- Anforderungen:
  - Möglichkeit neue Todos über ein Form hinzuzufügen
  - Anzeigen von offenen / abgeschlossenen Todos
  - Möglichkeit Todo abzuhaken
  - Möglichkeit Todo zu löschen
  - Zeige mit einer Statistik die Anzahl der offenen / geschlossenen Todos
  - Optionales CSS styling hinzufügen
- Lösungsansatz: https://github.com/michischmidt/react-schulung

trinnovative.

7 - Packages & React Ecosystem - npm

- React wird lokal in Node.js Umgebung entwickelt
- Packages für Node.js: npmjs.com (public registry)
- Empfehlung:
  - JS Tool Manger: volta.sh ->
    Projektspezifisch Node.js / npm / yarn Versionen lokal ändern

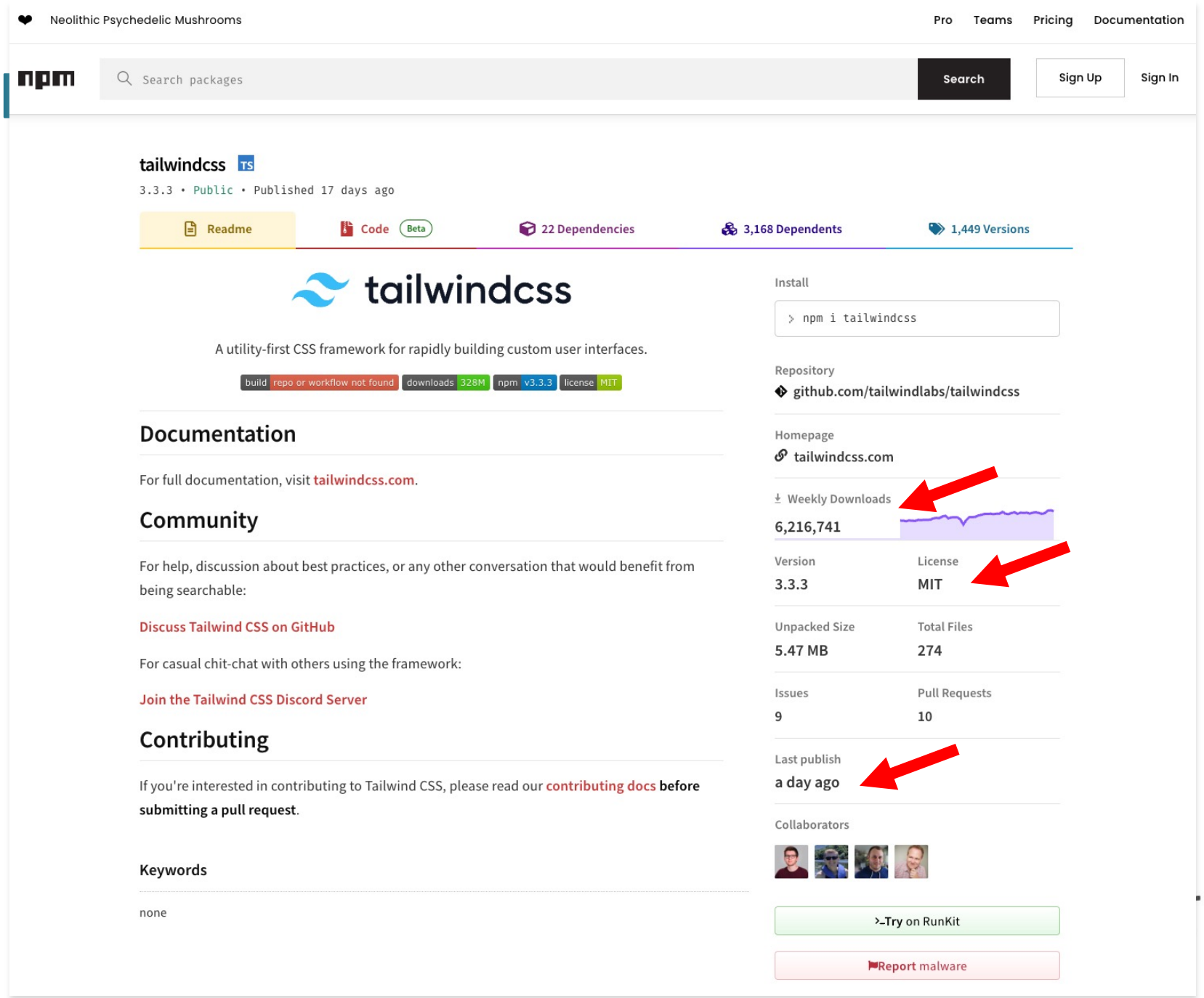
trinnovative.

7 - Packages & React Ecosystem - Ecosystem

- Frameworks
- UI Components
- Styling
- API Queries
- State Management
- Routing
- Forms
- Animation
- ...

trinnovative.

7 - Pac



innovative.

7 - Packages & React Ecosystem - Frameworks

- https://react.dev/learn/start-a-new-react-project
- Vite.js ✅
- Next.js ✅
- Remix
- Gatsby
- Expo (React Native Mobile) ✅
- (Preact)

trinnovative.

✅ persönlich mit gearbeitet

7 - Packages & React Ecosystem - UI Components & Styling

- **Style**: TailwindCSS (TW)
- **Styled TW components**: Headlessui
- **Components**: Material UI (MUI)
- **Animation**: Framer Motion, React Spring, AutoAnimate

trinnovative.

7 - Packages & React Ecosystem - API Queries

- **Http Client**: Axios, Fetch
- **Async State Management**: React(TanStack)-Query, React-SWR

7 - Packages & React Ecosystem - State Management

- Globaler State kann gewünscht sein (z.B. Suchleiste in Navbar):
- React-Query ✅
- Jotai ✅
- Zustand (✅)
- Redux
- MobX

trinnovative.

7 - Packages & React Ecosystem - Routing

- Next.js (via Folder Structure)
- Routing: React-Router
- (Beta): TanStack-Router

7 - Packages & React Ecosystem - Forms

- **Schema validation (TS):** Zod
- **Form**: React-Hook-Form, Formik

7 - Packages & React Ecosystem - Recap

- npm
- Frameworks
- UI Components / Styling
- API Queries
- State Management
- Routing
- Forms

trinnovative.

8 - API Abfragen - Abfragen in JS (then / catch)

- Promises: Handling Async Code
  - Mit async / await
  - Oder .then().catch(err)

```
import axios from 'axios';

axis.get('https://api.example.com/data’)
  .then(response => response.json())
  .then(data => {
    // Handle the data after it is resolved
    console.log(data);
  })
  .catch(error => {
    // Handle the error if the Promise is rejected
    console.error(error);
  });
```

nnovative.

8 - API Abfragen - Abfragen in JS (async / await)

- Beispiel async / await

```
import axios from 'axios';

async function getUser() {
  try {
    const response = await axios.get('/user?ID=123’);
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

trinnovative.

8 - API Abfragen - In React

```
export const MyComponent = () => {
  const [data, setData] = useState(null);

  const loadData = async () => {
    const newData = await fetchData();
    setData(newData);
  };

  return (
    <button onClick={loadData}>
      Load some data from an API
    </button>
  )
}
```

trinnovative.

7 - API Abfragen - In React mit React-Query

- Caching previous responses for reuse (globally)
- Tracking loading status (success / loading / error)
- Automatically sending a query:
  - When component is first mounted
  - When query parameter changes
  - On window clickback
  - ...

React Query

trinnovative.

8 - API Abfragen - GET React-Query

```
export const MyComponent = () => {
  const [searchTerm, setSearchTerm] = useState('');

  const { status, data } = useQuery({
    queryKey: ['search', searchTerm],
    queryFn: () => axios.get(`https://api/search/${searchTerm}`),
  });

  return (
    <div>
      <h1>{searchTerm}</h1>
      <input onChange={setSearchTerm} />
    </div>
  )
}
```

trinnovative.

8 - API Abfragen - GET React-Query

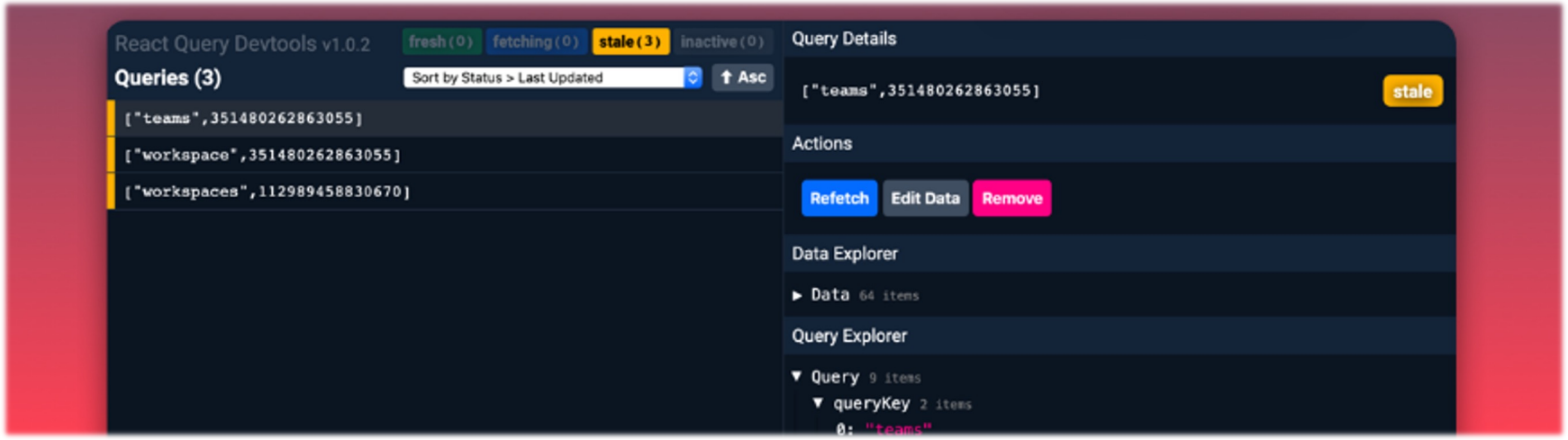
```
// In TicketComponent.tsx
const { data: ticket, isLoading: ticketIsLoading } = useGetTicket(ticketKey);
```

```
// GET In ticket-service.ts
export const useGetTicket = (IssueKey?: string) =>
  useQuery<Ticket, Error>(
  ["ticket", IssueKey],
  async () => {
    const response = await axios.get(`https://ticket/${IssueKey}`);
    return response.data;
  },
  { enabled: !!IssueKey }
);
```

trinnovative.

8 - API Abfragen - React-Query

- Muss noch nach der Installation aufgesetzt werden
  https://tanstack.com/query/v3/docs/react/quick-start
- Devtools können auch eingebunden werden
  https://tanstack.com/query/v3/docs/react/devtools



trinnovative.

8 - API Abfragen - Mutations React-Query

- PUT, POST, PATCH, DELETE
- Können auch durch Nutzer Interaktionen ausgelöst werden

trinnovative.

8 - API Abfragen - Mutations React-Query

```
// In ticket-service.ts
export const useUpdateTickets = () => {
  const queryClient = useQueryClient();

return useMutation(
  async ({
    chassisId,
    ticketDto,
  }: {
    chassisId: string;
    ticketDto: TicketDTO;
  }): Promise<void> => {
    await client.post(`JiraTicket/PostTicket/${chassisId}`, ticketDto);
    toast.success("Ticket erfolgreich erstellt.");

    await queryClient.invalidateQueries({ queryKey: ["tickets", chassisId] });
  }
 );
};
```

8 - API Abfragen - Recap

- JS Promises
- Abfragen in React
- React Query
- GET & Mutations

trinnovative.

9 - Side Effects- Basics

- Wenn Components Props / State sich ändern oder die Component das erste mal gemounted wird
- “Main Effect”: Component (re)-renders mit den aktuellen Daten
- Potenzielle “Side Effects”:
  - Triggern von API Queries
  - Speichern von Daten in die Browser Storage
  - Timer starten
  - …

trinnovative.

9 - Side Effects- Pure Components

- Keep Components “Pure” https://react.dev/learn/keeping-components-pure#side-effects-unintended-consequences
- Pure Components
  - Should not change any objects / variables that exist before rendering
  - Same Inputs, Same Output
  - Rendering should not depend on each others rendering sequence
  - Strive to express logic in JSX. Use event handler and (potentially) useEffect to “change things”

trinnovative.

9 - Side Effects- Effect Hook

- https://react.dev/reference/react/useEffect
- "Lets you synchronize a component with an external system"

```
useEffect(() => {
  // Effect
  first;

  // Cleanup (Optional)
  return () => {
    second;
  };

// Dependencies (Optional)
}, [third]);
```

trinnovative.

9 - Side Effects- Effect Hook

- **First**: Auszuführende Action
- **Second**: Cleanup Function
  - Wird vor dem nächsten Run des Effects und vor dem unmounten der Component ausgeführt
- **Third**: Dependencies (Array)
  - Re-(renders) Component wenn sich Dep. ändern

```
useEffect(() => {
  // Effect
  first;

  // Cleanup (Optional)
  return () => {
    second;
  };

// Dependencies (Optional)
}, [third]);
```

trinnovative.

8 - Side Effects- Effect Hook (Wechselkurs Beispiel)

```
const [from, setFrom] = useState("USD");
const [to, setTo] = useState("EUR");
const [rate, setRate] = useState(null);

async function loadRate(from, to) {
  const data = await currencyRates(from, to);
  setRate(data.rate);
}

useEffect(() => {
  loadRate(from, to);
}, [from, to]);
```

trinnovative.

9 - Side Effects- Effect Hook ohne Dependencies

- Ohne den Dependency Parameter wird der Effect bei jedem Rendering ausgeführt
- Benutzt um z.B. veraltete Daten zu vermeiden

```
useEffect(() => {
  ensureExchangeRateIsLoaded();
});
```

trinnovative.

9 - Side Effects- Recap

- Basics
- Pure Components
- Effect Hook (useEffect)

10 - **Context** - Basics

- Erlaubt es geschachtelten Components das lesen und subscriben von Values
- Ohne das explizite Props Passing nach unten
- Interface of Context erlaubt es für Daten & Event Handlers
- Component höher im Component Tree **provides** Context
- Components tiefer **consumen** diesen
- **Aber Components die consumen, Rerendern wenn sich ein beliebiger State im Context ändern**

trinnovative.

10 - **Context** - Anwendungsfälle

- ThemeContext
- LanguageContext
- UserContext
- …

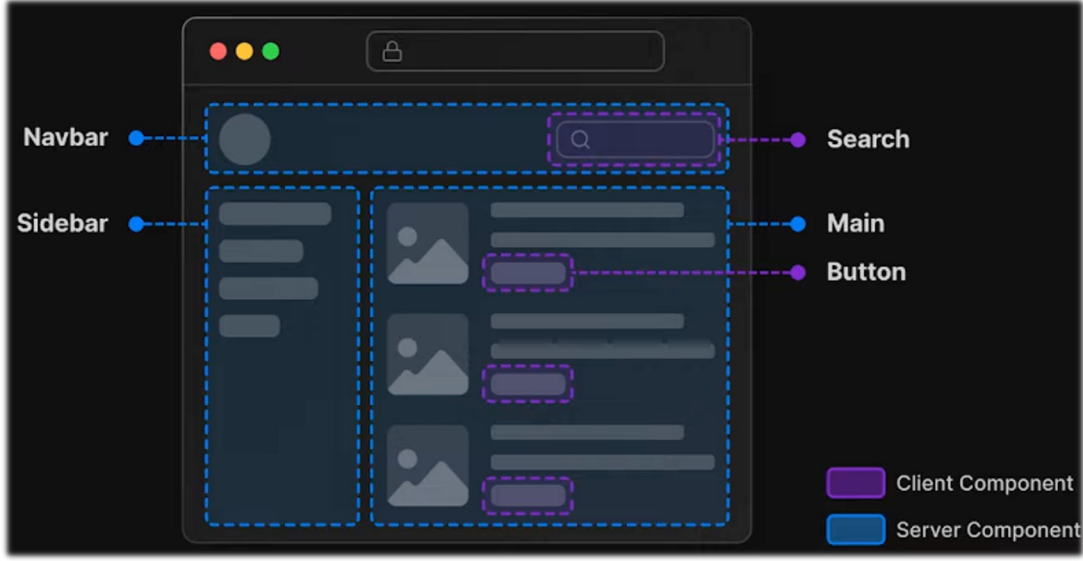
```
<LanguageProvider>
  <ThemeProvider>
    <App>
      <TodoList>
        <TodoItem />
      </TodoList>
    </App>
  </ThemeProvider>
</LanguageProvider>
```

trinnovative.

10 - **Context** - Recap

- Basics
- Anwendungsfälle

9 - **Ausblick** - in die Zukunft



Next.js SSR

- **S**erver-**S**ide **R**endering
  - React Server Components
- Next.js, Remix, …
- Shift von statisch generierter “App” auf Webserver
- Zu dynamisch serverseitig gerenderten Seiten / Components

  -> Welche an Client / Browser ausgeliefert werden können

  (https://vercel.com/blog/understanding-react-server-components)

trinnovative.

Zusammenfassung - React

- Was macht React aus?
  - **JSX, States, (Custom) Components, (Custom) Hooks, Event Handler, Rendering, …**
- Wie denke ich als React Entwickler?
  - **Component basierend, Modular, “Pure“, …**
- Wie und mit was setze ich mein React Projekt auf?
  - **Entwicklungsumgebung, Framework, Libraries, …**

Zusammenfassung - React

- Wie Frage ich Daten ab um diese Dynamisch anzuzeigen?
  - **Data fetching, React Query, …**
- Welche weitern Tools habe in React?
  - **Side Effects, Context, …**
- Rat fürs Entwickeln: Macht einfach! 👀
  - **Schnelles entwickeln und iterieren möglich**
  - **Dankbare Dokumentationen & Community**
  - **Und ihr könnt nichts kaputt machen**

12 - **Hands on** - State of the Art Todo App 💡

- Anforderungen:
  - Vite.js & TypeScript
  - Todo App mit bisherigen Anforderungen
  - Components / Styling mit Material UI (MUI)
  - Daten Abfragen über https://dummyjson.com/ (/todos)
  - API Quering mit React Query (GET und Mutations)
  - (Gerne noch erweitern mit eigenen Featuren!)

trinnovative.

8 - API Abfragen - Mutations React-Query

```
import { useMutation, useQueryClient } from 'react-query';
import { apiAddTodo } from './todosApi';
// ...
const queryClient = useQueryClient();

function invalidateTodos() {
  queryClient.invalidateQueries({ queryKey: ['todos'] });
}

const addTodoMutation = useMutation({
  mutationFn: apiAddTodo,
  onSuccess: invalidateTodos,
});

// START: Funktion wird durch submit form aufgerufen
function handleAddTodo(newTitle) {
  addTodoMutation.mutate({ title: newTitle });
}
```

vative.