1. Fundamental of React

# 1.1 Hello React

-**SPA** (Single-page application) made it easier to build web app that advanced beyond vanilla JS and JQueryy. **React** was released by Facebook in **2013**.

-In the past, websites were **rendered** from **server**: user visits URL in browser and requests one HTML file and associated files. After network delay, users see the rendered HTML in browser (client).

-Modern JS shifted the focus from **server** to **client**. A user visits URL and request one small HTML file and one larger JS file. After network delay, user see the by JS rendered HTML in browser. Additional page transition wouldn’t request more files from web server, but would use initially requested JS to render new page. Every additional interaction by user is handled on client. In this modern version, the server delivers mainly JavaScript across the wire with one minimal HTML file. HTML file then executes all linked JS from files on client-side to render entire app.

-SPA is one bulk of JS, which is organized in folders and files, to create app whereas SPA framework (React) gives all tools to architect it. React takes over for rendering everything in browser as HTML and for dealing with user interactions with JS.

-How we moved from websites to web app: https://www.robinwieruch.de/web-applications/

# 1.2 Requirements

-**Editor** and **Terminal**: Visual Studio Code

-**VCS**: Github

<https://www.robinwieruch.de/git-essential-commands/>

-**Node** and **NPM**

+Both are used to manage libraries (node packages) you will need. These node packages can be libraries or whole frameworks. We install external node packages via npm

+node –version and npm –version.

<https://www.robinwieruch.de/npm-crash-course/>

-**Yarn** and **pnpm**: <https://yarnpkg.com/> https://pnpm.io/

# 1.3 Setting up a React Project

-We use **Vite** to **set up React application**: vite.dev

+Vite is a modern build tool for status quo web frameworks which comes with sensible defaults while staying highly extensible for specific use cases (SVG support, Lint support, TypeScript)

+The essential core of Vite is development server ( start React app on local machine) and a bundler (outputs optimized files for production ready deployment)

-There are 2 ways to create project with Vite: online template (vite.dev/guide/#trying-vite-online) and creating a React project with Viton on local machine for working on it in IDE/editor.

-Crash course for navigating on command line:

A white background with black text

AI-generated content may be incorrect.

**-Create React project** hacker-stories



-**Install** all 3rd party **dependencies** of project and run it locally:

A white sign with black text

AI-generated content may be incorrect.

# 1.4 Project Structure

-code .: see project structure

A close up of a document

AI-generated content may be incorrect.

A text on a white background

AI-generated content may be incorrect.

-src/App.jsx used to implement React components. Later you want to split up React components into multiple files, each file maintains one or more components on its own.

-src/main.jsx is entry point to React world.

-src/index.css and src/App.css style overall app and components.

# 1.5 npm Scripts

-All project-specific commands can be found in package.json under scripts:

A screenshot of a computer code

AI-generated content may be incorrect.

+these script are executed with **npm run <script>:**

A black text on a white background

AI-generated content may be incorrect.

-preview: run production-ready build on local machine for testing purposes

+npm run build -> npm run preview

# 1.6 Meet the React Project

-Every React app is built on the foundation of React components. The 1st React component which is located in src/App.jsx.

-This file will be our focus throughout this book.

-Reduce the component to a more lightweight version

A screenshot of a computer program

AI-generated content may be incorrect.

-Optionally you can make src/index.css and src/App.css file bank for starting from a clean state style-wise.

-App component is just a JS function, it’s defined in PascalCase. Component has to start with a capital letter. App component is a function component: the modern way of using component in React.

-A function component can have implementation details between function signature and return statement:

A screenshot of a computer code

AI-generated content may be incorrect.

-Variable defined in function’s body will re-defined each time this function runs.

A screenshot of a computer code

AI-generated content may be incorrect.

-The function of a component runs every time a component is displayed in browser. This happens for the initial rendering of component, but also whenever the component updates because it has to display sth in different due to changes (re-rendering)

-Since we don’t want to re-define a variable within a function every time the function runs, we could define this variable outside of components as well.

A screenshot of a computer code

AI-generated content may be incorrect.

# 1.7 React JSX

-JSX (JavaScript XML): combine HTML and JS

A screenshot of a computer code

AI-generated content may be incorrect.

-The bridge between React and development server is React Fast Refresh (prior to that it was React Hot Loader) on React’s side and Hot Module Replacement on development server’s side.

-HTML input field and HTML label:

A screenshot of a computer program

AI-generated content may be incorrect.

+htmlFor reflects the for attribute in HTML. JPX replaces a handful of internal HTML attributes caused by internal implementation details of React.

-Find all supported HTML attributes in React documentation: [DOM Elements – React](https://legacy.reactjs.org/docs/dom-elements.html#all-supported-html-attributes)

-React use camelCase naming convention: [Camel case - Wikipedia](https://en.wikipedia.org/wiki/Camel_case)

-When using HTML in JSX, React translate all HTML attributes to JS where certain words such as class or for are reserved during rendering process. Therefore React came up with replacement like className and htmlFor. Once the actual HTML is rendered for browser, the attributes get translated back to their native variant.

A diagram of a programming language

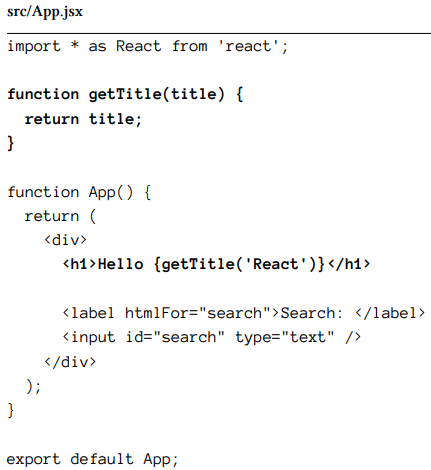
AI-generated content may be incorrect.

-Access properties within JSX:

A screenshot of a computer code

AI-generated content may be incorrect.

-While HTML can be used almost (except for the attributes) in its native way in JSX, everything in curly braces can be used to interpolate JS in it.



-The underlying build tools can be configured to acknowledge JSX in .js file. If they are configured this way, they will transpile JSX to JS. Tools like Vite embrace the .jsx extension though.

A screen shot of a computer code

AI-generated content may be incorrect.

-JSX is the favorite things when being asked about React. Without any extra templating syntax (except curly brace), we can use JS in HTML. Every JS data structure can be used within HTML with JSX.

# 1.8 Linting with ESlint (Optional)

-Linting: process in programming where code is analyzed for potential errors, bugs, and style issues.

-ESLint is popular linting tool for JS.

-Install the respective plugin:

A close-up of a plugin

AI-generated content may be incorrect.

-vite.config.js: allow us to customize the development and build process of a Vite-based project. It gives us options such as setting public path, configure plugins, and modify the build ouput.

A screenshot of a computer code

AI-generated content may be incorrect.

-Install the ESlint dependency: 

-Install one of ESLint’s many standardized linting configurations for React project:

A close up of a sign

AI-generated content may be incorrect.

-If you start project on cmd again, you will see error:



-Therefore we will create ESLint configuration file to define our linting rules: 

A screenshot of a computer code

AI-generated content may be incorrect.

-When starting app on cmd, you will see warning:

A black text on a white background

AI-generated content may be incorrect.

-In VSCode, you can install ESLint Extension.

# 1.9. Lists in React

-When working with JS, most often data is array of objects. **map()**: iterate over each item of a list in order to return a new version of each item

A screenshot of a computer code

AI-generated content may be incorrect.

-Render each object with its title property in React by map() in JSX:

A screen shot of a computer code

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

-Without any made up templating syntax, it’s possible to use JS to map from a list of items to a list of HTML elements. That’s what JSX is for developer in the end: just JS mixed with HTML

A diagram of a diagram

AI-generated content may be incorrect.

-key is an HTML attribute and should be a stable identifier:

A computer screen shot of a code

AI-generated content may be incorrect.

+The **key attribute** used for one specific reason: whenever React has to re-render a list, it checks whether an item has changed. When using keys, React can exchange the changed item.

A close-up of some words

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

+Using index should be avoided though, because it comes with the same rendering performance issues, it can cause actual bugs in UI when order of items got changed.

A computer screen shot of a program

AI-generated content may be incorrect.

# 1.10 Meet another React Component

-Instead of making one component larger and more complex, we’ll split one component into multiple components. We will start with a new List component which extracts functionalities from App component:

A screen shot of a computer program

AI-generated content may be incorrect.

-List component can be used in App:

A screenshot of a computer code

AI-generated content may be incorrect.

-Create Search component:

A screenshot of a computer code

AI-generated content may be incorrect.A black text on a white background

AI-generated content may be incorrect.

-React app consists of many hierarchical components

A diagram of a child development

AI-generated content may be incorrect.

# 1.11 React Component Instantiation

-A class is most often used in OOP languages. JS as a multi-paradigm programming language allows functional programming and OOP to co-exist side-by-side.

A computer screen shot of a code

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.A close-up of a computer code

AI-generated content may be incorrect.

-The concept of JS class with declaration and instantiation is similar to React component, which also has only one component declaration, but can have multiple component instances:

A screenshot of a computer program

AI-generated content may be incorrect.

-One we defined component, we can use it as an element in JSX. The element produces an instance of component. You can create as many instances of a component as you want as long as you have a component declaration. It’s not much different from JS class declare + instantiate. But technically JS class and React component are not the same.

# 1.12 React DOM

-src/main.jsx: see App components instantiation with <App /> element

A screen shot of a computer code

AI-generated content may be incorrect.

+React DOM is usually once in React app to hook React into native HTML world.

+Open index.html and spot HTML element where id=”root”: this is the element where React inserts itself into HTML to bootstrap React app-starting with App component

+createRoot(): instantiate React

+render(): represent the entry point component (root component). Normally the entry point component is the instance of App component, but it can be any other JSX too:

A screen shot of a computer code

AI-generated content may be incorrect.

-React DOM is everything that’s needed to integrate React into any website which uses HTML. If you start a React app from scratch, there’s only one ReactDom.createRoo(). In a legacy app that used sth else than React before, see multiple ReactDOM.createRoot() calls, because only certain areas of app are powered by React.

# 1.13 React Component Declaration

-We have used the standard function declaration, though arrow function can be used more concisely and therefore can establish a new standard for declaring function components

A screen shot of a computer code

AI-generated content may be incorrect.A white background with black text

AI-generated content may be incorrect.

-Not only function components can be reactored, other functions like callback function:

A screenshot of a computer code

AI-generated content may be incorrect.

-Block body (curly braces), concise body: implicit return statement:

A screenshot of a computer code

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

-Often block bodies will be necessary to introduce more business logic between function signature and return statement:

A screenshot of a computer code

AI-generated content may be incorrect.

# 1.14 Handler Function in JSX

-In native HTML, we can add event handlers by addEventListener() on DOM node.

-In React, refactor component function to block body and define a function for the change event of input field (event handler). Then the function can passed to onChange() (JSX named attribute) on HTML input field

A screen shot of a computer program

AI-generated content may be incorrect.

-Synthetic event: logging occur as JS object and input field’s internal value

+ React synthetic event is a wrapper around browser’s native event.

-React as library for SPA, there was need for enhanced functionalities on the event to prevent native browser behavior.

+Example: native HTML submit a form triggered a page fresh. In React this page refresh should be prevented.

-Always pass function to handlers, not the return value of function, except when the return value is a function

A screenshot of a computer code

AI-generated content may be incorrect.

# 1.15 React Props

-**Global scope**: it isn’t maintainable with multiple variables across multiple components.

-**props**:pass variables from one component to another

-list HTML attribute:

A screenshot of a computer code

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

-Retrieve the list:

A screenshot of a computer code

AI-generated content may be incorrect.

+Everything that we pass from a parent component to a child component via component element’s HTML attribute can be accessed in child component. The child component receives a parameter (props) as object in its function signature which includes all the passed attributes as properties

-Another use case for React props is List component and its child component: perform the component extraction and pass each item along to List component’s new child component.

+Example: extract an Item component from List component and pass each item in map()

A computer screen shot of a program

AI-generated content may be incorrect.

# 1.16 React State

-React state introduces a mutable data structure. These stateful values get instantiated in a React component as co called state, can be passed with props as vehicle down to child components, but can also get mutated by using a function to modify state. When a state gets mutated, the component with state and all child components will re-render

A diagram of a product

AI-generated content may be incorrect.

+**Props** used to **pass information** down the component hierarchy. **State** used to **change information** over time.

+Example: when user types text into HTML input field in Search, he wants to see this information (state) displayed next to it.

+**useState()**: tell React that we want to have a **stateful value** which **changes over time**. Whenever this stateful value changes, the affected component will **re-render** to use it.

A computer code with text

AI-generated content may be incorrect.

+useState() takes an **initial state** as an argument. Call this method will return an array with 2 entries: 1st entry represents **current state**. 2nd entry is a function to update this state. The book will refer to this function as **state updater function**. -> display the current state (read) and to update it (write)

A diagram of a function

AI-generated content may be incorrect.

+When user types into input field, input field’s change event runs into event handler. Handler’s logic uses event’s value of target and state updater function to set updated state. After that, the component re-renders (the component function runs). The updated state becomes current state and displayed.

-The initial rendering happens when React component get displayed in browser. When a side-effect occur (type into input field), the change is captured in React state which forces a re-rendering of all components affected by this change; meaning the component which manages state and all its descendant components

A diagram of state

AI-generated content may be incorrect.

-useState() is called React hook. It’s only one of many hooks provided by React and this section only scratched the surface of hooks in React.

-When UI is re-rendered because of a state change, useState() hook uses the most recent state from its internal closure ([JavaScript Closure by Example](https://www.robinwieruch.de/javascript-closure/)). Next to each component React allocates an object where information like state is stored in memory. The memory gets cleaned up once a component is not rendered anymore through JS garbage collection.

# 1. 17 Callback Handlers in JSX

-While props are passed down as information from parent to child component, state can be used to change information over time. Using props as vehicle to transport information, we only talk to descendant components. Using state, we can make information stateful, but this information can only be passed down by using props as container.

-We want to use the state somewhere else:

+Example: use state in App component to filter stories by searchTerm before passed to List component.

A diagram of a diagram of a person's face

AI-generated content may be incorrect.

+Props are only passed downwards. We can use **callback handler**: event handler (A) is passed as function in props to another component (B), is executed there as callback handler (C), and calls back to the place it was introduced (D):

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

+When user types into input field now, the function that is passed down from App to Search runs. We can notify App when user types into input field in Search. Callback handler becomes implicit vehicle to communicate upwards the component tree.

A drawing of a diagram

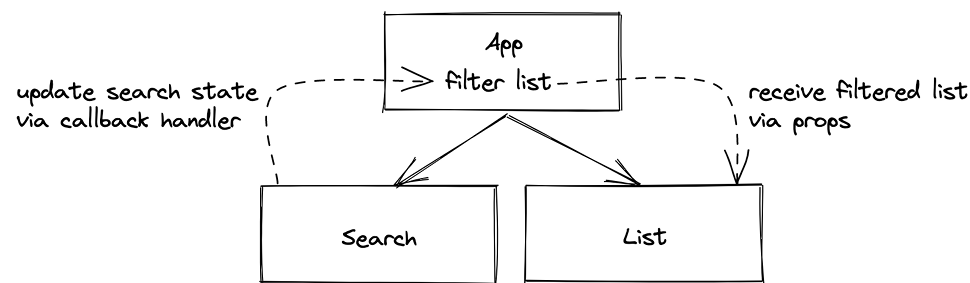
AI-generated content may be incorrect.

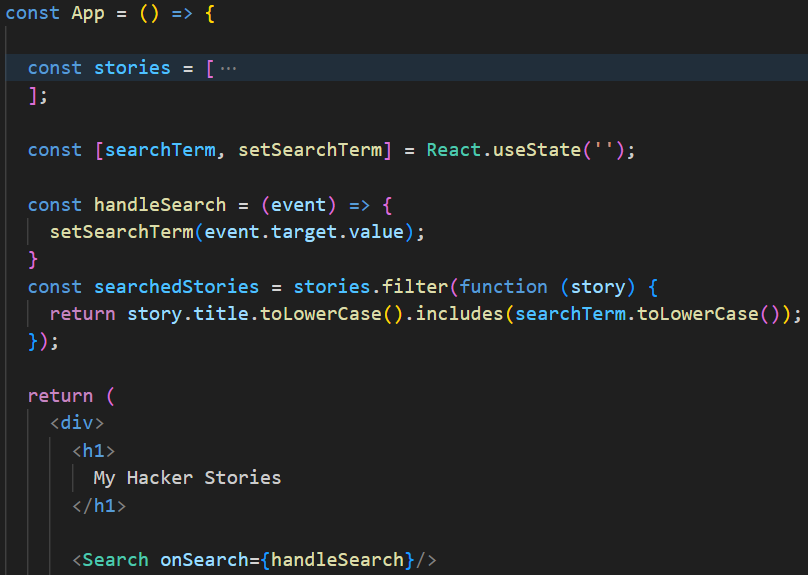
+The **concept of callback handler** in a nutshell: pass a function from a parent component to a child component via props; we can this function in the child component, but have the actual implementation of called function in parent component. In other words, when an event handler is passed as props from parent component to its child component, it becomes a callback handler.

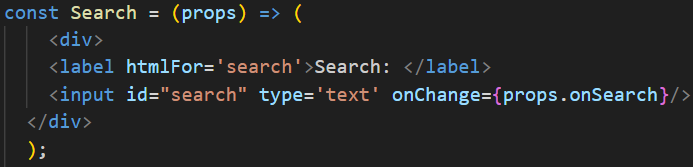
# 1.18 Lifting State in React

-Move the state from Search component to App component, pass the state updater function to Search component as callback handler and use it to update the state when user types into input field. Then use new state in App component to filter() stories before passed to List component

-**Lifting state:** moving state from one component to another

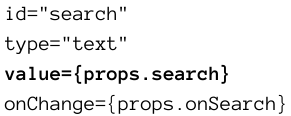
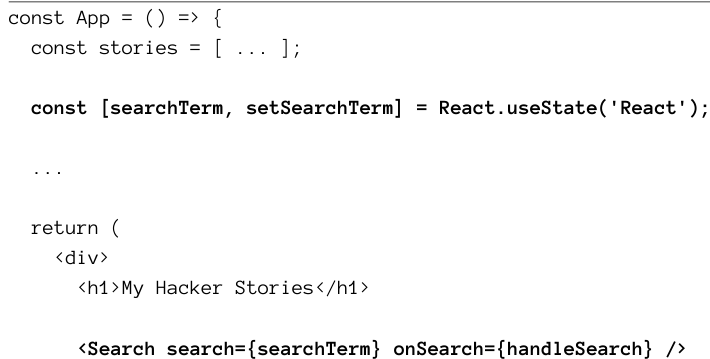




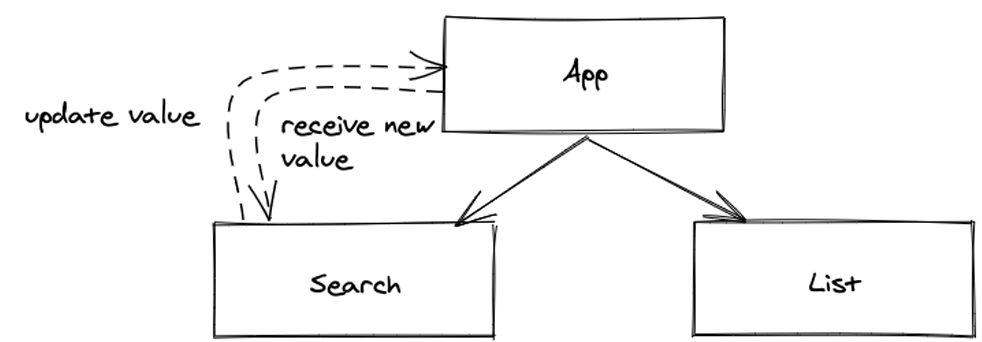


# 1.19 React Controlled Components

-Try use ‘React’ as initial state instead of empty string. HTML element should know about React stat, we provide the current state as **value** to it:



+Instead of giving HTML element the freedom of keeping track of its internal state, it uses React’s state by leveraging element’s value attribute. When HTML element emits a change event, new value is written to Reacts state and re-renders component. Then HTML element uses recent state as value



+Input field became **controlled element**, Search component became **controlled component**.

# 1.20 Props Handling (Advanced)

-Props are passed from parent to child down to component tree. It’s useful to know a few tricks to make passing props more convenient

A diagram of a application

AI-generated content may be incorrect.

**1.21.1 Props Destructing via Object Destructuring**

-**JS object destructuring: https**://mzl.la/30KbXTC

A screenshot of a computer code

AI-generated content may be incorrect.A close up of words

AI-generated content may be incorrect.

-Make use of **props** **destructuring**: refactor Search component’s arrow function from body into block body. Apply destructuring of props object in component function body:

A computer screen shot of a code

AI-generated content may be incorrect.

+That’s a basic destructuring of props object in React component, so that object’s properties can be used in component.

+We can destruct props objects **in function signature**, omit function’s block body of component:

A screenshot of a computer code

AI-generated content may be incorrect.

+Then we can access all information without dealing with props container. List and Item components can perform same props destructuring:

A computer screen shot of a code

AI-generated content may be incorrect.A computer code with black text

AI-generated content may be incorrect.

**1.21.2 Nested Destructuring**

-Even though item object has been destructured from props in Item component’s function signature, it isn’t directly used in Item component. item object only passes its information to elements.

-**Nested destructuring**:

A screen shot of a computer code

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

+**Nested destructuring** helps us to **gather all needed information** of item object in function signature for its immediate usage in component’s element. It can still be useful in other scenarios:

A screenshot of a computer program

AI-generated content may be incorrect.

1.21.3 Spread and Rest Operators

-Rather passing item as an object from List to Item component, we **pass every property** of item object:

A computer code with black text

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

+Now even though Item component function signature is more concise, the clutter ended up in List component instead, because every property is passed to Item component.

-We can improve by **JS spread operator**: https://mzl.la/3jetIkn

A screen shot of a computer code

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

+JS spread operator allows us to spreadl all key/value pairs of an object to another object. This can be done in React’s JSX. Instead of passing each property once at a time via props from List to Item component, we cause JS spread operator to pass all object’s key /value pairs as attribute/value pairs to JSX element:

A computer code with black text

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

+This refactoring made the process of passing information from List to Item component more concise.

-**JS rest destructuring**: always at the **last part** of an object **destructuring**

A white background with red text

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

+Use in List component to separate objectID:

A computer code with black text

AI-generated content may be incorrect.

+Rest operator happens on the left side, the spread operator happens on the right side. The rest operator used to separate an object from some of its properties

-Rules of thumb:

A close up of text

AI-generated content may be incorrect.

# 1.21 React Side-Effects

-React component returned output is defined by its props and state. **Side-effects** can affect this output too, they are used to interact with **3rd party APIs** (localStorage, remote APIs for data fetching), with HTML elements for width and height, or with built-in JS functions (timers, intervals).

-**Make Search remember recent search**: use a side-effect to **store** **recent search** from browser’s **local storage** and retrieve it upon initial component initialization.

A close-up of a computer code

AI-generated content may be incorrect.

+Use stored value or default value:

A close up of a word

AI-generated content may be incorrect.

-If we use **setSearchTerm** state updater function **somewhere else** in app, we break feature because **local storage doesn’t get updated**. Let’s fix by handling side-effect at a centralized place and not in a specific handler. We use React’s **useEffect** Hook to **trigger** the **desired side-effect** **each time searchTerm change**:

A white background with black text

AI-generated content may be incorrect.

A white background with black dots

AI-generated content may be incorrect.

+1st arg: function that runs our side-effect. 2nd arg: a dependency array of variables. If one of these variables changes, function of side-effect is called. If dependency array doesn’t have, function for side-effect run on every render. If dependency array is empty array, function for side-effect is called once when the component renders for the 1st time.

-After all, **useEffect()** can be triggered when component is 1st mounted, but also if one of its values (state, props, derived values from state/props) is updated.

# 1.22 React Custom Hooks (Advanced)

-useState(): for values that change over time, useEffect(): used to opt into lifecycle of your component to introduce side-effects

-React custom Hooks: create a hook

A computer code with black text

AI-generated content may be incorrect.A close up of a text

AI-generated content may be incorrect.

# 1.23 React Fragments

-**React Fragment**:

A screenshot of a computer code

AI-generated content may be incorrect.

+A fragment **wraps sibling elements** into a **single top-level element** **without** **adding** them to **rendered output**.

+A more popular alternative is using fragments in **shorthand version**

A screenshot of a computer code

AI-generated content may be incorrect.

# 1.24 Reusable React Component

-Since Search component doesn’t have any actual search functionality, we can make component reuseable for the rest of app: Pass a **dynamic id and label prop** to Search component, rename actual value and callback handler to sth more generic, and rename the component:

A screenshot of a computer code

AI-generated content may be incorrect.

A computer code with black text

AI-generated content may be incorrect.

A black text on a white background

AI-generated content may be incorrect.

-If we want to support numbers or phone numbers, type attribute needs to be accessible from outside:

A screenshot of a computer program

AI-generated content may be incorrect.

-More Reusable React Components:

https://www.robinwieruch.de/react-button/

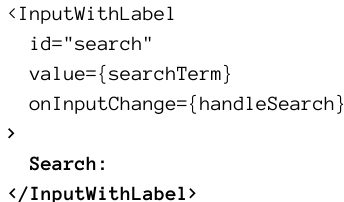
<https://www.robinwieruch.de/react-radio-button/>

<https://www.robinwieruch.de/react-checkbox/>

https://www.robinwieruch.de/react-dropdown/

# 1.25 React Component Composition

-Until now we only used components as self-closing tags. **Component composition**: opening and closing tag instead for React elements



-Component composition is one of React’s more powerful features. **Instead of using label prop** before, we inserted **text “Search”** between element’s tags. You have access to this information via **React’s children prop** to render everything.



-Everything that’s passed between component’s elements can be accessed as children in component and be rendered somewhere. Sometimes, you want to have more freedom from outside regarding what to render on inside of component:

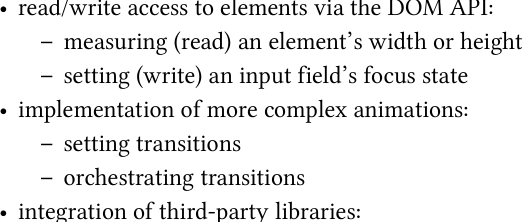


-Read more: https://www.robinwieruch.de/react-component-composition/

# 1.26 Imperative React

-React is inherently **declarative:** tell React what elements you want to see, now how to create these

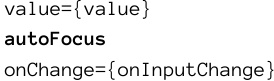
-However, there are cases when we won’t want everything to be declarative: like imperative access to rendered elements (most often side-effect)



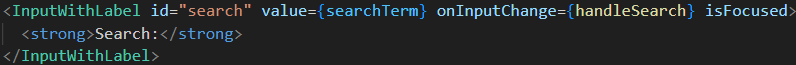


-Example: set the focus of input field

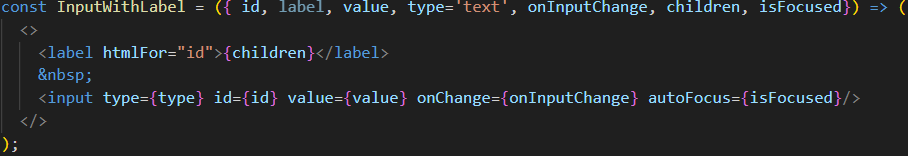
-**Declarative way**: set autoFocus



+If App component renders 2 components, only the last rendered component receives this feature. We can pass a prop **isFocused** which lets developer decide whether the input field should have autoFocus



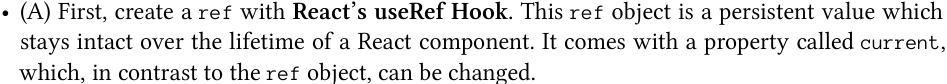
+isFocused equivalent to isFocused={true}

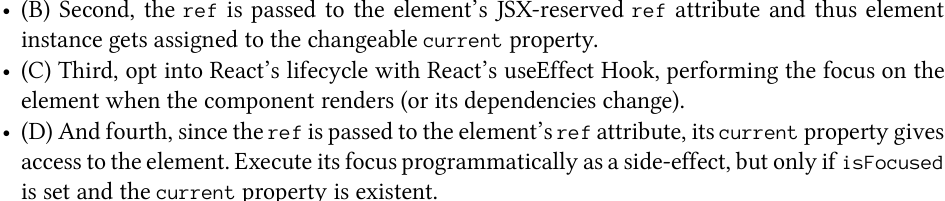


-**Imperative way**: execute focus() programmatically on input field’s element via DOM API once it has been rendered

A screen shot of a computer code

AI-generated content may be incorrect.





-Read more:

+refs: <https://www.robinwieruch.de/react-ref/>

+Create Slider with ref: <https://www.robinwieruch.de/react-slider/>

+Create image from React component with ref

<https://www.robinwieruch.de/react-component-to-image/>

+Create custom hook with ref:

https://www.robinwieruch.de/react-custom-hook-check-if-overflow/

+Why frameworks matter: https://www.robinwieruch.de/why-frameworks-matter/

# 1.27 Inline Handler in JSX

-Remove items from list:

+useState()



+Write event handler:

A white background with black text

AI-generated content may be incorrect.



+Pass to Item:

A computer code with black text

AI-generated content may be incorrect.

+Item use callback handler in a new handler:

A computer screen shot of a code

AI-generated content may be incorrect.

-inline handler: execute the callback handler function in component right in JSX. 2 solutions to use onRemoveItem() in Item component as inline handler

+**JS’s bind method**: bind arguments directly to that function that should be used to execute.



+Use **inline arrow function**:



Inline arrow may difficult to debug because JS logic may be hidden in JSX. You can use **block body**:

A computer code with text

AI-generated content may be incorrect.

-Read more:

+Add, update, remove items in list:

<https://www.robinwieruch.de/react-add-item-to-list>

<https://www.robinwieruch.de/react-update-item-in-list/>

<https://www.robinwieruch.de/react-remove-item-from-list>

+Computed properties in React:

<https://www.robinwieruch.de/react-computed-properties/>

+Review handlers, callback handlers, inline handlers

<https://www.robinwieruch.de/react-event-handler/>

# 1.28 React Asynchronous Data

-Data from remote backend/database arrives asynchronously for client-side apps. We must render a component before initiating the data fetching.

-Start with a **function** that returns a **promise** with **data** in its shorthand version once it resolves. The resolved object **holds the list of stories**:

A white background with black text

AI-generated content may be incorrect.

-Start off with an **empty list** of stories and simulate **fetching stories asynchronously**. **useEffect()** call function and resolve returned promise as a side-effect. Empty dependency array make side-effect only runs once the component renders for 1st time.

A white background with black text

AI-generated content may be incorrect.

-Even though data should arrive asynchronously when starting app, it appears to arrive synchronously, as it’s rendered immediately. Git it a bit of a realistic delay, as every network request to a remote API would come with a delay:

+Remove shorthand version of promise

A black and white text

AI-generated content may be incorrect.

+delay it for 2 seconds:

A white background with black text

AI-generated content may be incorrect.

-Read more:

+JS Promises: <https://mzl.la/3aTGuQz>

+Faking a remote API with JS: <https://www.robinwieruch.de/javascript-fake-api/>

Use mock data in React: <https://www.robinwieruch.de/react-mock-data/>

# 1.29 React Conditional Rendering

-User would see feedback (e.g. loading spinner) when data gets loaded.

-Conditional rendering: always happen if we have to render different JSX based on information (e.g. state, props).

-Dealing with asynchronous data is a good use case to make use of conditional rendering: For example, when app initializes 1st time, there is no data to start with. Next, we are loading data and get data to display, or data fetching fails and we get an error.

-A few of these cases are already taken care of: As the initial state is [] -> app don’t break when we filter and map over the list.

-Loading state which will show loading indicator as feedback about pending data request:

+Introduce a new stateful value and set the state accordingly when data gets fetched

A screen shot of a computer program

AI-generated content may be incorrect.

A close-up of words

AI-generated content may be incorrect.

+Show user the loading indicator. A straightforward approach would be:

A screen shot of a computer code

AI-generated content may be incorrect.

However, this way only the loading indicator would render and nothing else. We can use **ternary operator** and produce a **conditional rendering** in JSX:

A screenshot of a computer program

AI-generated content may be incorrect.

-Introduce another state for error handling and handle it in promise’s catch()

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

+Give user feedback when sth wrong with another conditional rendering. It’s either rendering sth or nothing -> use &&

A white background with black text

AI-generated content may be incorrect.

-Read more:

+Conditional rendering: https://www.robinwieruch.de/conditional-rendering-react/

# 1.30 React Advanced State

-All state management in app makes heavy use of React’s useState hook. **useReducer** Hook enables ones to use more sophisticated state management for complex state structures and transitions.

-Reducer function always receives a **state** and an **action** and return a new state:

A screenshot of a computer code

AI-generated content may be incorrect.

+**action** is always **associated** with a **type** and as a best practice with a **payload**.

-useReducer() to manage stories: argument is initial state and return an array with 2 items: **current state** + state updater function (or **dispatch function**)

A computer code with black text

AI-generated content may be incorrect.

+The **dispatch function** can used instead of **setStories()**, it sets the state implicitly by dispatching an action for reducer. The action comes with a type and an optional payload

A screen shot of a computer code

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.

+A reducer and useReducer() hook are managing the state for stories. It’s valid to move this logic into reducer function and manage reducer with an action, which is another case for moving from imperative to declarative programming.

A screenshot of a computer program

AI-generated content may be incorrect.

+Now reducer function has to cover this new case in a new conditional state transition. If the condition to remove is set, reducer has all implementation details needed to remove. The action gives all necessary information (item’s identifier) to remove

A computer code with black text

AI-generated content may be incorrect.

-**Summary**: The reducer covers 2 state transitions: shows how to compute the current state and action into a new state, and uses some business logic for a state transition.

-Read more:

+Reducer and useReducer(): https://www.robinwieruch.de/react-usereducer-hook/

# 1.31 React Impossible States

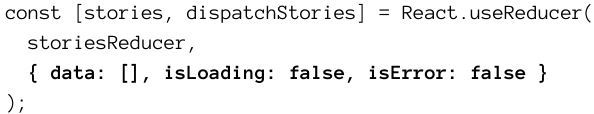
-All states related to asynchronous data belong together, which doesn’t only include stories as actual data, but also their loading and error states.

-Impossible state happens when an occurs for asynchronous data. Be wary once you see multiple state updater functions in a row.

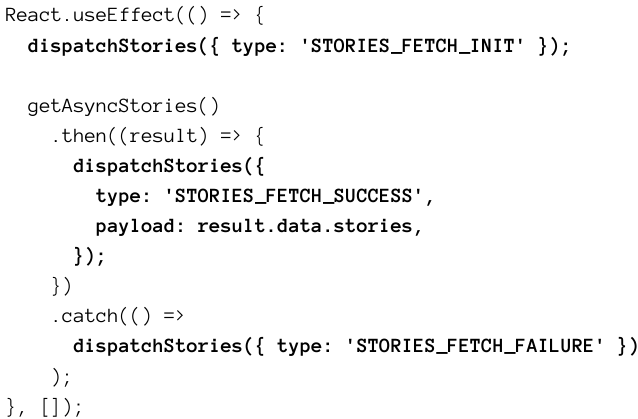
-We can improve chances of not dealing with such bugs by moving states that belong together from multiple useState (and useRender) hooks in a single userReducer hook:



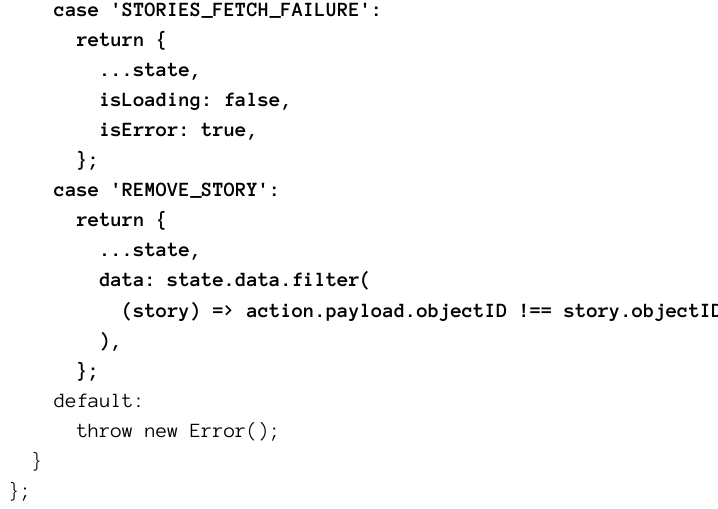
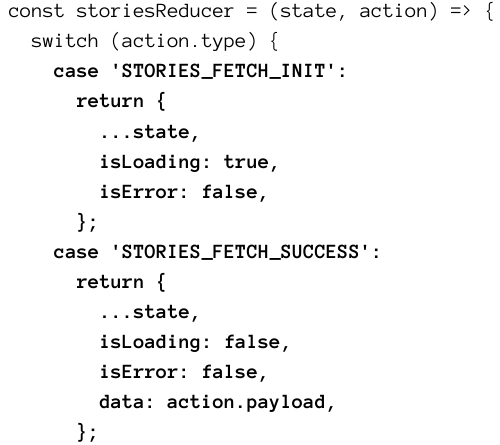
+Merge them into one useReducer hook for a unified state management



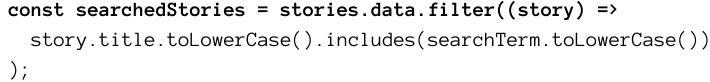
-We can’t use state updater function from useState() anymore, everything related to asynchronous data fetching must use the new **dispatch function** for the state transition. Use dispatch function receives a distinct **type** and **payload**:

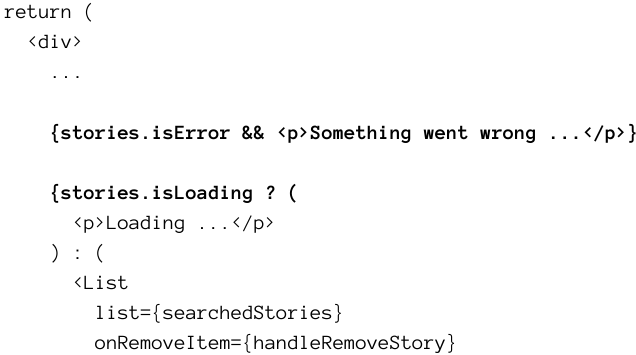


-**Reducer function**: For every transition, we return a **new state object** which contains all **key/value pairs** from **current state** object (via JS’s **spread operator**) and **new overwriting properties**:



+In REMOVE\_STORY, the state is a complex object with data, isLoading and isError. We need to **address** the **state** as **object**:





-**Test** with erroneous data fetching function:



-Read more

+When to use useState or useReducer:

<https://www.robinwieruch.de/react-usereducer-vs-usestate/>

+Deriving state from props

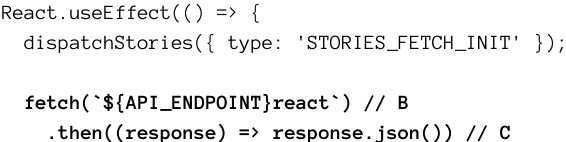
https://www.robinwieruch.de/react-derive-state-props/

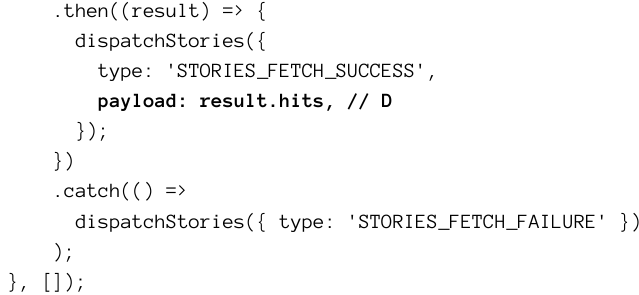
# 1.32 Data Fetching with React

-API: <https://hn.algolia.com/api>

-Fetch data directly:







+B: use native browser fetch API (<https://mzl.la/2Z1kyjU>)

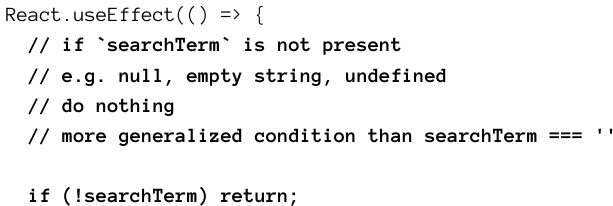
+C: response translated into JSON

+We use JS’s Template Literals for string interpolation: https://mzl.la/3jlcVfn



# 1.33 Data Re-Fetching in React

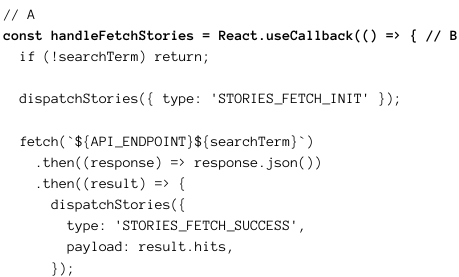




# 1.34 Memoized Functions in React (Advanced)

-Functions defined in components are mostly event handlers. As component is just a function, you can declare functions, function expressions and arrow function expressions in component. We can use memorized function by useCallback() Hook.

-Move data fetching logic to arrow function, expression (A), wrap this new function to useCallback (B), and call it in useEffect (C)

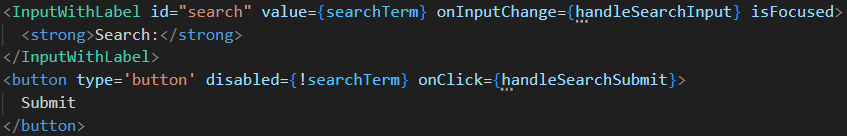


+We made data fetching logic as a function -> reusability and can used by other parts of app.

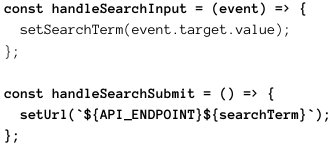
-Read more: https://www.robinwieruch.de/react-usecallback-hook/

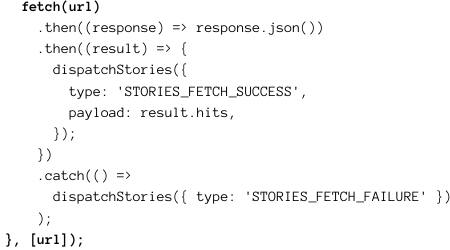
# 1.35 Explicit Data Fetching with React

-Refetch data only if user clicks confirmation button







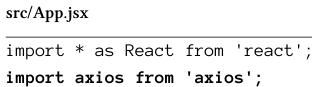
  
-Now searchTerm used just for updating input field’s state, and url introduced for triggering side-effect that fetches data only when user clicks button.

# 1.36 Third-Party Libraries in React

-There are ways to make fetch work in older browsers (<https://bit.ly/3ASC86Y>) and in tests (isomorphic fetch). One alternative is to substitute the **native fetch API** with a stable library like **axios**: perform asynchronous requests to remote APIs.

**-Install + import:**





-Axios **wraps result** into **a data object in JS**:



+You can use other HTTP methods: post()

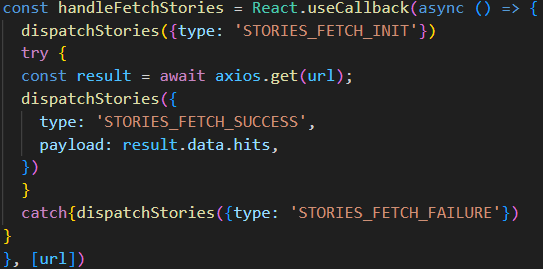
-Read more

+Popular libraries: <https://www.robinwieruch.de/react-libraries/>

+Axios: https://bit.ly/3jjEupg

# 1.37 Async/Await in React

-Remote API gives your frontend data, you need to work with data asynchronously. We have started to resolve promises with then/catch, but in modern JS we use **async/await** (<https://mzl.la/3AWyWaw>)



+To use async/await, our function requires async. Once you start using await on returned promises, everything reads like asynchronous code. Actions after await are note executed until promise resolves.

+Use try/catch to include error handling.

-Read more

+Helper library which simplifies error handling

https://bit.ly/3SBwGOT

# 1.38 Form in React