

An introduction to React

A large, faint, light-blue watermark of the React logo is centered in the background of the slide. The logo consists of a stylized 'R' formed by two interlocking circles.

What to expect

- A 6-8 hour interactive workshop
- Learning about React while building something with it
- Using modern React (functional components)
- Code examples and try-it-yourself challenges
- Feel free to ask questions

What not to expect

- Full coverage of every React feature (time)
- Server-side or react-native code (mobile)
- In-depth look at frameworks and libraries



Questions?

Let's build a filter-app together!

Example: Final product (end goal)

Food

Animals

Sports

Filter

Reset

Type something

type

☐ fruit

☐ vegetable

☐ root

taste

☐ sweet

☐ sour


☐ umami


☐ spicy


price


5


7 Items


 Apple


 Banana

 Lemon

 Tomato

 Ginger

 Strawberry

 Cherry

Why React?

- An industry standard
- Well documented, matured library
- Many 3rd party packages & tools
- Reusable components
- State management

Quick setup with Vite

1.

```
npm create vite@latest
```

2.

Need to install the following packages:

create-vite@4.4.1

Ok to proceed? (y) y

✓ Project name: ... vite-project

? Select a framework: › - Use arrow-keys. Return to submit.

Vanilla

Vue

› React

Preact

Lit

Svelte

Solid

Qwik

Others

? Select a variant: › - Use arrow-keys. Return to submit.

TypeScript

› TypeScript + SWC

JavaScript

JavaScript + SWC

3.

```
npm install
```

```
npm run dev
```

Where to start?

- Start with a single component
- Make it return basic & static HTML
- Future steps:
 - Make it dynamic (components & properties)
 - Make it interactive (state & event-handlers)

HTML structure within .jsx

- Conditionals
- Iterations
- Fragments

Conditionals

```
<div>
  {isLoading && <p>Loading...</p>}
</div>
```

```
<div>
  {isLoading ? <p>Loading...</p> : <Filter />}
</div>
```

- JavaScript expressions can be put inside JSX by placing it between {}
- There are many strategies for conditional JSX

Iterations

```
{options.map((option) => (  
  <li key={option.id}>  
    {option.id}  
  </li>  
))}
```

Arrays can be iterated over and each return more JSX

Fragments

```
<>
  <h1>Hello</h1>
  <p>World</p>
</>
```



```
<Fragment>
  <h1>Hello</h1>
  <p>World</p>
</Fragment>
```



```
<div>
  <h1>Hello</h1>
  <p>World</p>
</div>
```



- React requires single top-level elements
- Fragments group items without affecting the DOM (<div> could affect CSS & JS selectors)

food

animals

sports

Filter

Reset

Type something

type

- ☐ fruit
- ☐ vegetable
- ☐ root

taste

- ☐ sweet
- ☐ sour
- ☐ umami
- ☐ spicy

price





5

7 Items

 Apple


 Banana

 Lemon

 Tomato

 Ginger

 Strawberry

 Cherry

Split code into components

- Reusable
- Configurable
- Testable
- Readable

foodanimalssports

Tabs

FilterSettings

Filter

Reset

Textsearch

Type something

type

Checkboxgroup

☐ fruit☐ vegetable☐ root

taste

Checkboxgroup

☐ sweet☐ sour☐ umami☐ spicy


price


RangeSlider


5


7 Items


FilterResults


 Apple


 Banana

 Lemon

 Tomato

 Ginger

 Strawberry

 Cherry

Dynamic components

Example: Splitting into components

foodanimalssports

Filter

Reset

Type something

type

Checkboxgroup

☐ fruit☐ vegetable☐ root


taste


Checkboxgroup


☐ sweet☐ sour☐ umami☐ spicy


price


7 Items


 Apple

 Banana

 Lemon

 Tomato

 Ginger

 Strawberry

Component properties

- Make components re-usable
- Hands over state to parent components

Component properties

```
<Checkboxgroup
  label="taste"
  options={["sweet", "sour", "umami"]}
/>
```

```
<Checkboxgroup
  label="type"
  options={["fruit", "vegetables"]}
/>
```

Example: Component configuration
components/FilterSettings.tsx

```
export function Checkboxgroup({ label, options }) {
  return (
    <>
      <h3>{label}</h3>
      <ul>
        {options.map((option) => (
          <li key={option}>
            <label>
              <input
                name={`_${label}_checkboxgroup`}
                type="checkbox"
                value={option}
              />
              {option}
            </label>
          </li>
        ))}
      </ul>
    </>
  );
}
```

Example: Component configuration
components/Checkboxgroup.tsx

Recap

- Built static HTML with `.jsx`
 - Used iteration and conditionals for HTML generation
- Separated it into reusable React components
- Made components dynamic with properties
- Visually our app looks complete, but of course it is missing...



Interactivity

one of React's strengths

User input



For now let's focus on events which get triggered by direct user-input

Examples: click, focus, input, scroll, ..

Event-handling

```
document.querySelector('#search').addEventListener('change', (e) => {  
  updateFilterResults(e.target.value);  
});
```



The JS `addEventListener` doesn't work well:

- React's virtual DOM frequently recreates nodes
- Every render creates additional event-handlers (need to be cleaned up)

React event-handling

```
<input
  type="search"
  value={value}
  onChange={(event) => { /* do something */ }}
/>
```

Example: Final Product
components/TextSearch.tsx

```
<button onClick={(event) => /* do something */}>
  Reset
</button>
```

Example: Final Product
components/FilterReset.tsx

- Attach events directly inside .jsx
- Event-handlers change state, which is defined outside of .jsx

Interactivity checklist

- Create a variable to hold state
- Use the state in our component (JSX & JS)
- Update state on user-input (event)
- Re-render the app after state updates



State management

one more of React's strengths

useState()

```
const [activeTab, setActiveTab] = useState(0);  
const [search, setSearch] = useState("");
```

```
<input type="search"  
  value={search}  
  onChange={e => setSearch(e.target.value)}  
/>
```

- Returns a state variable and a setter function
- State is saved between renders
- React will re-render when state changes



Interactivity checklist



Create a variable to hold state

```
const [search, setSearch] = useState("");
```



Use the state in our component (JSX & JS)

```
<input type="search" value={search} onChange={e => setSearch(e.target.value)} />
```



Update state on user-input (event)

```
<input type="search" value={search} onChange={e => setSearch(e.target.value)} />
```



Re-render the app after state updates

```
// React re-renders automatically after state mutates!
```

Interactive <Tabs>

Example: Interactive <Tabs>

foodanimalssports

Tabs

Filter

Reset

Type something

type

☐ fruit

☐ vegetable

☐ root

taste

☐ sweet

☐ sour


☐ umami


☐ spicy


price


5


7 Items


 Apple


 Banana

 Lemon

 Tomato

 Ginger

 Strawberry

 Cherry

Interactive <Tabs>

```
import { useState } from "react";
import { Tabs } from "../components/Tabs";

export function ExampleInteractiveTabs() {
  const options = ["food", "animals", "sports"];
  const [activeFilter, setActiveFilter] =
    useState(options[0]);

  return (
    <>
      <Tabs
        options={options}
        active={activeFilter}
        onUpdate={setActiveFilter}
      />
      <pre>Checked tab: {activeFilter}</pre>
    </>
  );
}
```

Example: Interactive <Tabs>
InteractiveTabs.tsx

```
export function Tabs({ options, active, onUpdate }) {
  return (
    <div className="filter-navigation" role="tablist">
      {options.map((option) => (
        <button
          key={option}
          role="tab"
          aria-selected={active === option}
          className={`filter-navigation__button${
            active === option ? " active" : ""
          }}`
          onClick={() => onUpdate(option)}
        >
          {option}
        </button>
      ))}
    </div>
  );
}
```

Example: Interactive <Tabs>
components/Tabs.tsx

let, const, var

```
const search = ""  
  
<input type="search"  
  value={search}  
  onChange={e => { search = e.target.value }}  
/>
```



Wouldn't work because:

- The variable gets recreated on every render
- Updating the variable won't tell react to re-render

let, const, var

```
const [search, setSearch] = useState("");  
const searchLowerCase = search.toLowerCase();
```

- Not everything needs to be defined with `useState()`
- For example: derived variables
 - They won't need to be changed directly
 - Their origin will already cause a re-render on change

Lifting state up

- In React data always flows from the top to the bottom
- If state needs to be shared between adjacent components, move it to a parent and pass it down with properties
- Pass `setState` to a child so it can change the state of a parent, from where the updated state will flow down afterwards

Interactive <TextSearch>

Example: Interactive <TextSearch>

foodanimalssports

Filter

Reset

Textsearch

Type something

type

☐ fruit

☐ vegetable

☐ root

taste

☐ sweet

☐ sour


☐ umami


☐ spicy


price


5


7 Items


 Apple


 Banana

 Lemon

 Tomato

 Ginger

 Strawberry

 Cherry

Interactive <TextSearch>

```
import { useState } from "react";
import { Tabs } from "../components/Tabs";

export function ExampleTabs() {
  const options = ["food", "animals", "sports"];
  const [activeFilter, setActiveFilter] =
    useState(options[0]);

  return (
    <>
      <Tabs
        options={options}
        active={activeFilter}
        onUpdate={setActiveFilter}
      />
      <pre>Checked tab: {activeFilter}</pre>
    </>
  );
}
```

Work in progress

```
export function Tabs({ options, active, onUpdate }) {
  return (
    <div className="filter-navigation" role="tablist">
      {options.map((option) => (
        <button
          key={option}
          role="tab"
          aria-selected={active === option}
          className={`filter-navigation__button${
            active === option ? " active" : ""
          }}
          onClick={() => onUpdate(option)}
        >
          {option}
        </button>
      ))}
    </div>
  );
}
```

Interactive <Checkboxgroup>

Example: Interactive <Checkboxgroup>

food

animals

sports

Filter

Reset

Type something

type

Checkboxgroup

☐ fruit

☐ vegetable

☐ root

taste

Checkboxgroup

☐ sweet

☐ sour


☐ umami


☐ spicy


price


5


7 Items


 Apple


 Banana

 Lemon

 Tomato

 Ginger

 Strawberry

 Cherry

Interactive <Checkboxgroup>

```
import { useState } from "react";
import { Tabs } from "../components/Tabs";

export function ExampleTabs() {
  const options = ["food", "animals", "sports"];
  const [activeFilter, setActiveFilter] =
    useState(options[0]);

  return (
    <>
      <Tabs
        options={options}
        active={activeFilter}
        onUpdate={setActiveFilter}
      />
      <pre>Checked tab: {activeFilter}</pre>
    </>
  );
}
```

Work in progress

```
export function Tabs({ options, active, onUpdate }) {
  return (
    <div className="filter-navigation" role="tablist">
      {options.map((option) => (
        <button
          key={option}
          role="tab"
          aria-selected={active === option}
          className={`filter-navigation__button${
            active === option ? " active" : ""
          }`}
          onClick={() => onUpdate(option)}
        >
          {option}
        </button>
      ))}
    </div>
  );
}
```

Work in progress

Recap

Hooks

useState

useRef

useSyncExternalStore

useTransition

useCallback

useEffect

useInsertionEffect

useContext

useReducer

useImperativeHandle

useLayoutEffect

useDeferredValue

use

useId

useDebugValue

useMemo

Hooks

- Hook are functions that let you “hook” into React’s state
- They need to run:
 - inside a functional component
 - in the same order every render
 - unconditionally

Pure Functions

- Always produce same output with same input
- Have no side-effects
- Can itself have impure functions within it's own boundaries

Functional Components

Work in progress

- React can easily abort incomplete renders
- Results can be cached
- Components can be rendered on the server

Mutating state

Where to mutate state?

- Inside event handlers
- Inside `useEffect()`

Detecting state changes

Work in progress

We want to fetch our data when the app initializes and when the category is changed

- This should happen without user-input

useEffect()

Work in progress

- Run after the component has rendered
- Has a dependency array that needs to match the effects dependencies
- If dependencies are empty it only runs once
- The returned function runs when component unmounts

useEffect()

Work in progress

- Its purpose is to synchronize with external systems
- Examples:
 - Send analytics events
 - Synchronize with native APIs like window resize
 - Synchronize with an external API like a chatrooms

useEffect()

Work in progress

Can easily be misused and introduce unnecessary complexity and performance costs.

How to avoid prop drilling?

```
<div className="filter">
  <FilterSettings
    filters={filters}
    updateCategory={updateCategory}
    updateRange={updateRange}
    updateSearch={updateSearch}
    reset={reset}
  />
  <FilterResults
    items={filterResults}
    search={filters.search.value}
    reset={reset}
  />
</div>
```

Example: Final Product
Filter.tsx

```
<div className="filter__settings">
  <FilterSettingsToolbar reset={reset} />

  <TextSearch
    value={filters.search.value}
    onUpdate={(e) =>
      updateSearch(e.target.value)}
  />
  ...
</div>
```

Example: Final Product
FilterSettings.tsx

useContext()

```
export const FilterContext = createContext(null);
```

```
<FilterContext.Provider items={items} >  
  <FilterSettings />  
  <FilterResults />  
</FilterContext.Provider>
```

Example: Final Product
components/Filter.tsx

```
import { FilterContext } from './Filter.tsx'
```

```
items = useContext(FilterContext)
```

Example: Final Product
components/FilterSettings.tsx
& components/FilterResults.tsx

Provides a shared state to all child components which subscribe with the useContext hook

useContext()

Work in progress

- Components which don't subscribe don't have to know the data even exists
- Can lead to unintended re-renders

useRef()

Work in progress

- Changing `ref.current` doesn't cause a rerender!

Component updates

Work in progress

- React tries to only update parts of DOM which need to be updated
- React has it's own virtual DOM
- Sometimes we want to explicitly update components

key attribute

Work in progress

- Whenever the key attribute changes, React will treat it like a different element and cause a rerender.

How to handle complex state

Work in progress

There are multiple states all describing the state of the filter settings.

=> It would be good to unify them

useReducer()

Work in progress

UseReducer()

Similar to useState() but it can have multiple functions to update the state it holds

JS mutations

	copy	mutating
add	<code>.concat(), [...arr]</code>	<code>.push(), .unshift()</code>
remove	<code>.filter(), slice()</code>	<code>.pop(), .shift(), .splice()</code>
replace	<code>.map()</code>	<code>.splice(), arr[i] =</code>
sort	<code>[...arr] => arr.sort()</code>	<code>.reverse(), .sort()</code>

Source: <https://react.dev/learn/updating-arrays-in-state>

Copying deeply nested state

Copying nested state can become very tedious and hard to read:

```
case "SET_RANGE":  
  return {  
    ...state,  
    ranges: {  
      ...state.ranges,  
      [action.payload.id]: {  
        ...state.ranges[action.payload.id],  
        value: action.payload.value,  
      },  
    },  
  };
```

useFilter.tsx

Immer (Framework)

- Immer allows state to be mutated directly
 - Replace `useState()` with `useImmer()`
 - Replace `useReducer()` with `immerReducer()`

```
case "SET_RANGE":  
  return {  
    ...state,  
    ranges: {  
      ...state.ranges,  
      [action.payload.id]: {  
        ...state.ranges[action.payload.id],  
        value: action.payload.value,  
      },  
    },  
  };  
};
```



```
case "SET_RANGE":  
  state.ranges[action.payload.id].value =  
    action.payload.value;  
  break;
```

Custom hooks

Work in progress

- Need to start with use
- Only are considered hooks if they wrap around React hooks, otherwise they're just functions
- Are essentially just wrappers like functions



Performance issues

If we had complex filters with thousands of items, slower devices might run into performance issues

Filter

Reset

Type something

type


☐ fruit


☐ vegetable


taste


☐ sweet

1825 Items

 Apple

 Banana

 Lemon

 Tomato

Performance optimizations

Work in progress

React has hooks to optimize performance e.g.:

- `useMemo()`
- `useCallback()`
- `useTransition()`
- `useDeferredValue()`

useMemo()

Work in progress

```
const filterResults = useMemo(() => {  
  // costly filter operation  
  
  return filteredItems;  
}, [filters, items]);
```

- Cache the result of a heavy computation
- Not needed for simple calculations
- Only updates when dependency array changes

useCallback()

Work in progress

- Like useMemo() but it returns a function instead of a value
- Referential equality



TypeScript

Work in progress

State Management

Work in progress

- SWR
- React Query
- Redux & Redux Toolkit
- Zustand
- Jotai

Routing

Work in progress

- React-Router
- NextJS

Next steps & Ideas

- Add routing (detail views)
- Save the filter-state inside url query params for deep linking capabilities
- Extract Filter types into separate modules and make it extendible

Recommendations

Work in progress

- Official React.dev - Learn & Documentation
<https://react.dev/learn>
- Jack Herrington - Typescript & React
<https://www.youtube.com/watch?v=j8AVXNozac8>
- Web Dev Simplified – React Hooks Explained
<https://www.youtube.com/watch?v=O6P86uwfdR0>



Thank you!