

# Intermediate React

[tsevdos.me](https://tsevdos.me) / [@tsevdos](https://twitter.com/tsevdos)

# Agenda

All the content can be found [here](#).

- vite app
- hooks
- forms and events

# Rules

Feel free to interrupt me for:

- questions
- relevant comments

# Vite app

Vite is the easiest way to create client-side, single-page React applications. It offers a modern build setup with no configuration.

# Vite app init

```
npm create vite@latest my-app -- --template react
cd my-app
npm install
npm run dev
```

# Vite app

Files and directory structure (demo).

# Hooks

Let you put state and logic to your functional components!

# Hook rules

- only call hooks from React function components
- only call hooks at the top level
- don't call hooks inside loops, conditions, or nested functions
- custom hooks start with useSomething PascalCase function



# React Hooks

- useState
- useEffect
- useReducer
- useContext
- useCallback
- useMemo
- useRef

# React Hooks

- `useState`: is used to declare a state variable and can be initialized with any type of value. It returns an array with the (current) state and a function used to update the state.
- `useReducer`: An alternative to `useState`. Accepts a reducer of type `(state, action) => newState`, and returns the current state paired with a dispatch method.

# React Hooks

- `useEffect`: accepts an effect "action" as an anonymous function as the first argument. Skip applying an effect if certain values haven't changed between re-renders. To do so, pass an array as an optional second argument to `useEffect`. Finally, some effects might require cleanup so they return a function.
- `useCallback`: `useCallback` will return a memoized version of the callback that only changes if one of the inputs has changed.

# React Hooks

- `useMemo`: Returns a memoized value. Pass a "create" function and an array of inputs. `useMemo` will only recompute the memoized value when one of the inputs has changed.
- `useRef`: `useRef` returns a mutable ref object whose ".current" property is initialized to the passed argument (`initialValue`). The returned object will persist for the full lifetime of the component.

# useState

```
const [state, setState] = useState("my value");
```

# useEffect

```
// accepts two arguments: a function, and dependency array
useEffect(() => {
  // do stuff
  return () => {}; // run this return function when component unmounts
}, []); // dependency array
```

# useReducer

```
const [state, dispatch] = useReducer(reducer, initialState);
```

# Hooks

Examples.



# Forms and Events

- inputs events
- form events

# Components and events

- SyntheticEvent
- cross-browser wrapper around the browser's native event
- it has the same interface as the browser's native event, including `stopPropagation()` and `preventDefault()`
- you have access to the native event using `event.nativeEvent`

# Components and events

- react events are named using camelCase, rather than lowercase
- supported events

# Forms and Events

Examples.

# **Mini project: ToDo list**

Starting point (`./intermediate/workshop/todo-app`).

# Mini project: exercise 1

Create the add todo functionality.

# Mini project: exercise 2

Create the toggle todo functionality.

# Mini project: exercise 3

Create the delete todo functionality.



# Mini project: exercise 4

Replace the hard-coded todos with server data, using this [endpoint](#).

# Recap

- vite app
- hooks
- forms and events

# Recap: hooks

- state (useState / useReducer)
- useEffect

# Recap: forms and events

- (SyntheticEvent) event object
- input handling
- form handling

**That's all folks**

**Questions / Discussions?**