REACT

THE GOAL FOR THIS SESSION

1. *Be able to use events in React, just like you would in vanilla*
2. *Be able to create and modify simple state in React*
3. *Be able to analyze simple scenarios and decide when and where to put state*

AGENDA

1. events
2. state #1
   - `const [count, setCount] = useState(0)`
   - The React extension
   - example on how react "fails" (setState is asynchronous)

EVENTS

events in React are quite different from what you're used to

They actually look a lot like "stuff you've been told not to do"

```
1  export default function EventTester(props) {
2    function clicked(e) {
3      console.log("I was clicked");
4    }
5    return (
6      <article>
7        <button onClick={clicked}>Click me</button>
8      </article>
9    );
10 }
```

supported events

WARNING

LOTS of examples will do something like

```
<div onClick={}>Stuff</div>
```

Don't put click events on stuff like that, use buttons and anchors

And if you have no choice, research aria attributes

OTHER SYNTAXES FOR EVENTS

As with `addEventListener`, events in React receive a callback, and you'll see a lot of different syntaxes

So let's take a quick look

```
1  function A(props) {
2    function simple(evt) {
3      //do stuff
4    }
5    return <button onClick={simple}>Click me</button>;
6  }
7
8  function B(props) {
9    const arrow = (evt) => {
10     //do stuff
11   };
12   return <button onClick={arrow}>Click me</button>;
13 }
```

```
1 //These two are needed if we wish to pass custom arguments
2 //to the function
3
4 function C(props) {
5   function myFunc(greeting) {
6     //do stuff
7   }
8   return <button onClick={(e) => myFunc("Hi")}>Click me</button
9 }
```

```
1  //These two are needed if we wish to pass custom arguments
2  //to the function
3
4  function C(props) {
5    function myFunc(greeting) {
6      //do stuff
7    }
8    return (
9      <button
10       onClick={(e) => {
11         myFunc("Hi");
12       }}
13     >
14       Click me
15     </button>
```

STATE, #1

1. State is "the state of our application at any given point"
2. or some smaller part of our app, like
   - `let menuOpen = false;`
   - `let userSignedIn = true;`
   - `let blogPosts=[];`
   - `let valueInForm=42;`

THE REACT PROMISE

1. Whenever state changes, react will "re-render" anything dependent on that state
2. Whenever props change, react will "re-render" anything dependent on those props

- In practice, that means, that, whenever we change state, all components, that uses that state, either directly, or through props, will update automatically!!!
- Did the user log out? Just change a variable, and all components that knows about the user will update as needed

`props` are passed down from the parent

`state` lives inside a component

When either of these change, the affected components are "re-rendered"

(meaning the function runs again)

A "STATEFUL" COMPONENT

Stateful components consists of a few things

1. We import `useState`

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

2. `useState` is a function that, once called returns an array with two things, a state variable, and an "updater function"
3. When calling `useState` we pass the initial value for

4. We use the "updater function" to modify state to force a re-render
5. We use the "state variable" in our UI, as a condition, or simply data

Remember, every time "state" or "props" change, the UI is updated. So all we have to do is update state.....

It's quite strange initially, but SO powerful once you get it

useState, #1

INITIAL SETUP

```
1 import { useState } from "react";
2
3 export default function Counter() {
4   const initialCount = 0;
5   const [count, setCount] = useState(initialCount);
6 }
```

And we can use the `count` variable

```
1 import { useState } from "react";
2
3 export default function Counter() {
4   const initialCount = 0;
5   const [count, setCount] = useState(initialCount);
6   return <button>You've clicked me {count} times</button>;
7 }
```

`useState`, #2

- But the power comes from the "magic of React"
  1. When state changes, React updates (sorry)
  2. `useState` gives us an updater function
  3. In our case, we called it `setCount()`
- Calling our function will modify state, thereby forcing a re-render

useState, #3

- In `useState` we pass a function that:
  - receives the current state
  - Must return the new state

```
1 useState(function (prevState) {
2   return prevState + 1;
3 });
4
5 //or more commonly
6 useState((prevState) => prevState + 1);
```

"RULES OF HOOKS"

https://reactjs.org/docs/hooks-rules.html

Don't call Hooks inside loops, conditions, or nested functions

In general, only call `useState` in callbacks (events / useEffect)

useState, #5

```
1  import { useState } from "react";
2
3  export default function Counter() {
4    const initialCount = 0;
5    const [count, setCount] = useState(initialCount);
6    function handleClick() {
7      //set state to be equal to it's current value + 1
8      setCount((prevCount) => prevCount + 1);
9    }
10   return <button onClick={handleClick}>You've clicked me {count} times</
11 }
```

useState, #6

```
1  import { useState } from "react";
2
3  export default function Greeter() {
4    const [title, setTitle] = useState("Sir");
5    function handleClick() {
6      //just overwrite state
7      setTitle("Lord");
8    }
9    return <button onClick={handleClick}>Welcome {title} Jonas</b
10 }
```

Another form exists when the state is not based off a previous state

Let's look at Dev Tools

Repeat after me

I MUST NEVER MODIFY STATE DIRECTLY

I MUST NEVER MODIFY STATE DIRECTLY

I MUST NEVER MODIFY STATE DIRECTLY

I must never modify state directly

Always use the "updater" function

WHEN STATE FAILS

TODO: verify it fails

`useState` is asynchronous. React schedules the calls and handles them when it has the time

The following will work, but not correctly!

```
1  import { useState } from "react";
2  export default function App() {
3    const [count, setCount] = useState(0);
4    function increment() {
5      //this would be correct setCount(prevCount=>prevCount+1)
6      setCount(count + 1);
7      setCount(count + 1);
8      setCount(count + 1);
9      //or even worse: setCount(count++)
10   }
11   return <button onClick={increment}>{count}</button>;
12 }
```

MINI

ANIMAL

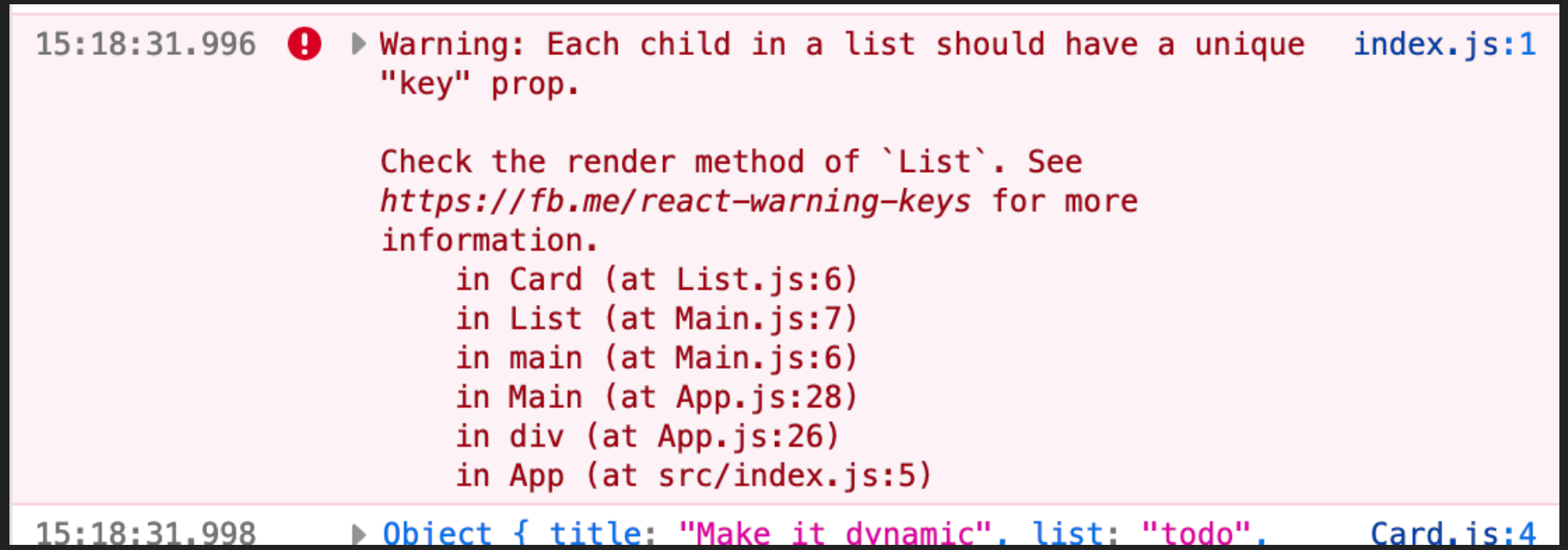
BASE

Let's solve it, quick and dirty

And record it

KEYS

Right now we have an error in the console



But basically it's just:

React needs a "key" so it can find the stuff to update fast

SOLUTION

1. Give each component that is "based" off an array a `key` property that is unique to that list
2. Something like `<card key={data.id} ...`

+3/4

Copy the mini animal base

If you can, try the following

1. Make each animal a component
2. Give each `<Animal />` it's own state (starred)
3. Add a star that is either dimmed or bright dependent on the state (starred)