# React 18: Time to upgrade!

Main Features, Key Differences, and Potential Dependency Incompatibility

accenture | Baltics

# Who I am?

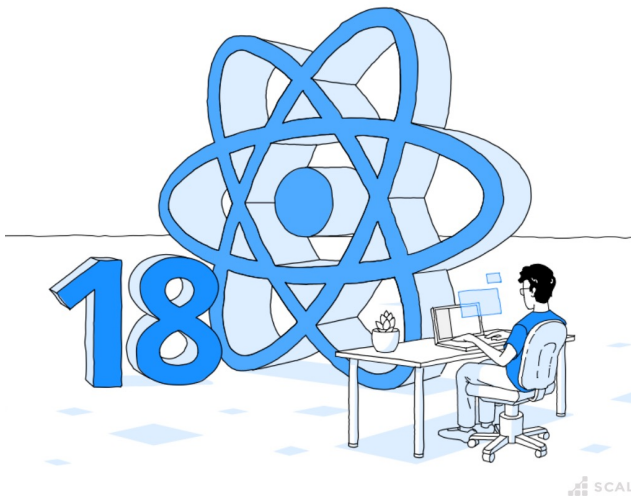
- Bachelor in Chemistry (including studies in Germany)
- Work experience in Chemistry and Industrial Production
- Music lover, traveller and happy dog owner
- Started SheGoesTech Javascript program in February 2021
- Currently Senior Analyst | Web Developer at Accenture Baltics

Katrina Merca

# Dev experience

- Work for UK based telco client.
- Mostly used technologies: ReactJS + TypeScript.
- Other technologies: Cypress, Jest/RTL, NodeJS etc.
- Contributions to clients internal libraries, documentation and innovation ideas, also hackathons.

# Agenda




**01** React 18 - short intro

**02** Breaking changes

**03** Key features

**04** Potential dependency incompatibilities

**05** Unit testing

**06** Migration tips & tricks

**07** Personal experience

**08** Q&A

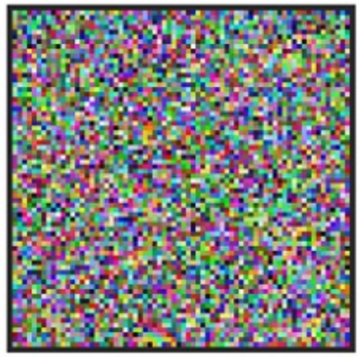
# React 18 – short intro

- Dropped in 2022.
- Can now adapt the rendering process to suit client devices.
- Now depends on modern browser features, including **Promise**, **Symbol**, and **Object.assign**.
- React no longer throws an error if you return **undefined** from a component.

- **Concurrent React** - new behind-the-scenes mechanism that enables React to prepare multiple versions of your UI at the same time.
- As a React developer, you focus on *what* you want the user experience to look like, and React handles *how* to deliver that experience.
- Key property of Concurrent React is that **rendering is interruptible**.
- Most of new features are built to take advantage of concurrent rendering.

- **React 18 is just the beginning of what is aimed to be built on this new foundation.**

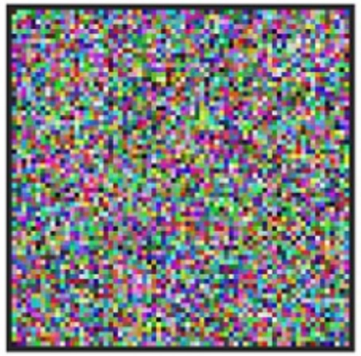
## Traditional or Block Rendering



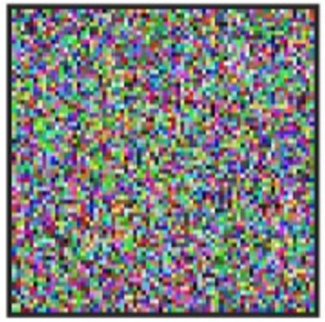
UI halts everytime user presses a key.

## Traditional or Block Rendering

aaas

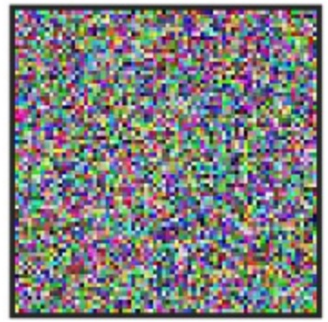


## Interruptible Rendering: Concurrent Mode



## Interruptible Rendering: Concurrent Mode

asdgfd



The User input box is updated with the new values that user is putting in even though the rest of the UI has not caught up.

# Breaking changes

## In most cases, React 18 upgrade shouldn’t really break your app, but:

### 1. Some of the new features might not work correctly if not fully migrated

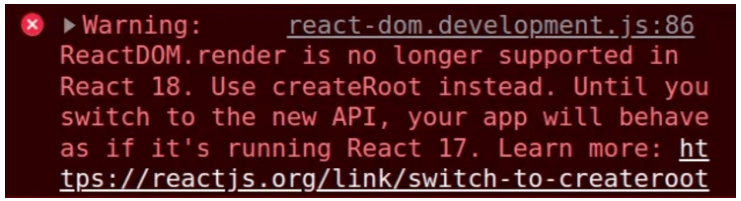
But you might probably get a bunch of warnings in terminal.

### 2. You will need to also upgrade all React related packages

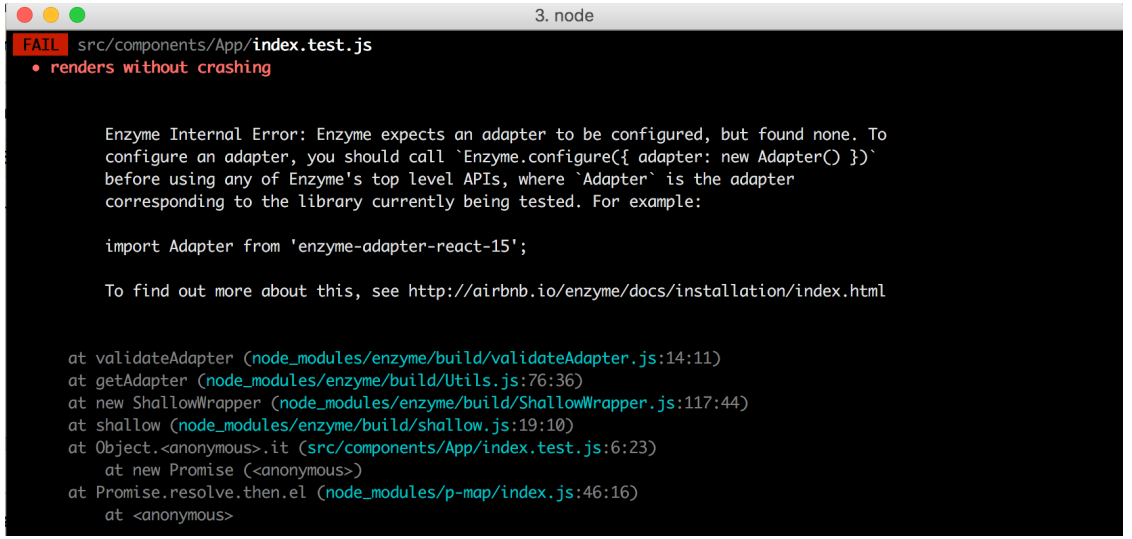
For example, react-dom, @types/react etc.

### 3. It’s possible that some of the external packages would not work with latest React version

For example, Enzyme for unit testing etc.







# Key features

## Quick guide

| Category | Feature |
| --- | --- |
| Concept | Concurrent React |
| Features | Automatic Batching, Transitions, Suspense on server |
| APIs | createRoot, hydrateRoot, renderToPipeableStream, renderToReadableStream |
| Hooks | useId, useTransition, useDeferredValue, useSyncExternalStore, useInsertionEffect |
| Updates | Strict mode |
| Deprecated/discouraged | ReactDOM.render, renderToString |

# Features > Automatic Batching

**Grouping multiple state updates into a single re-render for better performance.**

```
// Before: only React events were batched.
setTimeout(() => {
  setCount(c => c + 1);
  setFlag(f => !f);
  // React will render twice, once for each state update (no batching)
}, 1000);

// After: updates inside of timeouts, promises,
// native event handlers or any other event are batched.
setTimeout(() => {
  setCount(c => c + 1);
  setFlag(f => !f);
  // React will only re-render once at the end (that's batching!)
}, 1000);
```

# Features > Transitions

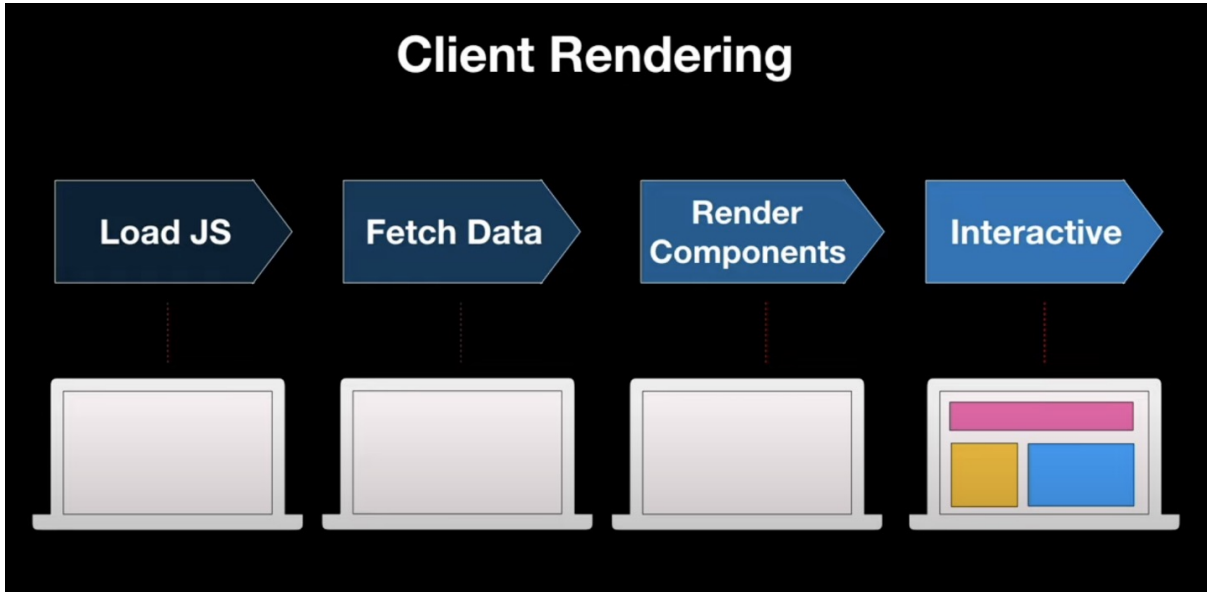
New concept to distinguish between urgent and non-urgent updates.

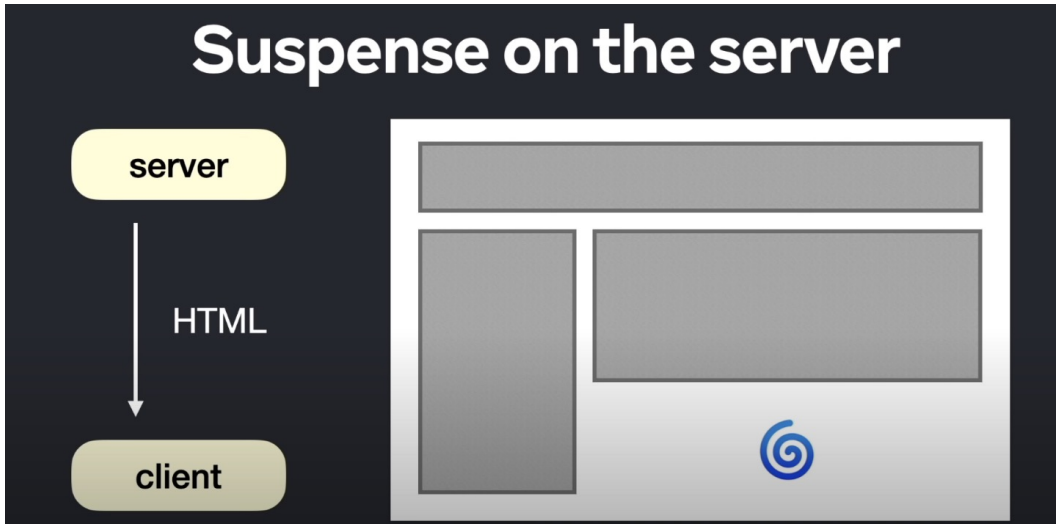
```
import { startTransition } from 'react';

// Urgent: Show what was typed
setInputValue(input);

// Mark any state updates inside as transitions
startTransition(() => {
  // Transition: Show the results
  setSearchQuery(input);
});
```

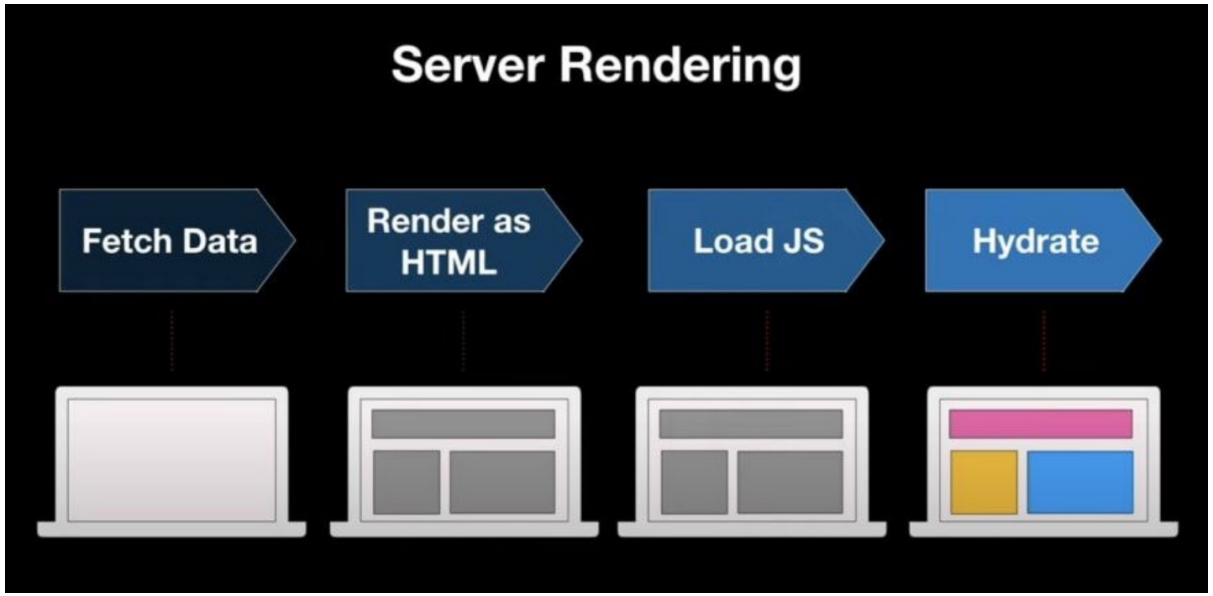
# Features > Suspense on server

**Now possible to have placeholder components, which will replace other components while they “suspend” or are being loaded.**

```
<Suspense fallback={<PageSkeleton />}>
  <RightColumn>
    <ProfileHeader />
  </RightColumn>
  <LeftColumn>
    <Suspense fallback={<LeftColumnSkeleton />}>
      <Comments />
      <Photos />
    </Suspense>
  </LeftColumn>
</Suspense>
```



# APIs > createRoot

**createRoot** is what actually enables new features of v18. If you use **render** as usually, app will also work as with previous versions. And in case of render you will see a deprecation note in your terminal.

```
// Before
import { render } from "react-dom";

const container = document.getElementById("app");
render(<App tab="home" />, container);

// After
import { createRoot } from "react-dom/client";

const container = document.getElementById("app");
const root = createRoot(container);
root.render(<App tab="home" />);
```

# Hooks > useID

**For generating unique IDs on both the client and server, while avoiding hydration mismatches.**

```
function NameFields() {
  const id = useId();
  return (
    <div>
      <label htmlFor={id + '-firstName'}>First Name</label>
      <div>
        <input id={id + '-firstName'} type="text" />
      </div>
      <label htmlFor={id + '-lastName'}>Last Name</label>
      <div>
        <input id={id + '-lastName'} type="text" />
      </div>
    </div>
  );
}
```

# Hooks > useTransition

**For additional control of transitions.**

```
function App() {
  const [isPending, startTransition] = useTransition();
  const [count, setCount] = useState(0);

  function handleClick() {
    startTransition(() => {
      setCount(c => c + 1);
    });
  }

  return (
    <div>
      {isPending && <Spinner />}
      <button onClick={handleClick}>{count}</button>
    </div>
  );
}
```

# Hooks > useDeferredValue

**For deferring of re-rendering for a non-urgent part of the tree.**

```
function Typeahead() {
  const query = useSearchQuery('');
  const deferredQuery = useDeferredValue(query);

  // Memoizing tells React to only re-render when deferredQuery changes,
  // not when query changes.
  const suggestions = useMemo(() =>
    <SearchSuggestions query={deferredQuery} />,
    [deferredQuery]
  );

  return (
    <>
      <SearchInput query={query} />
      <Suspense fallback="Loading results...">
        {suggestions}
      </Suspense>
    </>
  );
}
```

# Hooks > useSyncExternalStore

For reading and subscribing from external data sources in a way that's compatible with concurrent rendering features like selective hydration and time slicing.

```
const state = useSyncExternalStore(store.subscribe, store.getSnapshot);
```

```
const selectedField = useSyncExternalStore(
  store.subscribe,
  () => store.getSnapshot().selectedField,
);
```

# Hooks > useInsertionEffect

Meant to inject styles into the DOM before reading layout in useLayoutEffect. Identical to useEffect, but it fires synchronously before all DOM mutations.

```
useInsertionEffect(didUpdate);
```

# Updates > StrictMode

**<React.StrictMode>** is a development tool that helps developers test the robustness of their components as they transition to a future where mounting and unmounting will work with reusable state. Only available in development mode.

- Updated behaviour.
- **useEffect** might render twice (especially for data fetching).

Before v18:

```
* React mounts the component.
  * Layout effects are created.
  * Effects are created.
```

With v18:

```
* React mounts the component.
  * Layout effects are created.
  * Effects are created.
* React simulates unmounting the component.
  * Layout effects are destroyed.
  * Effects are destroyed.
* React simulates mounting the component with the previous state.
  * Layout effects are created.
  * Effects are created.
```

# Deprecated / discouraged

**ReactDOM.render -**
replaced with **ReactDOM.createRoot**

```
import ReactDOM from 'react-dom';
import App from 'App';

const container = document.getElementById('app');

ReactDOM.render(<App />, container);
```

```
import ReactDOM from 'react-dom';
import App from 'App';

const container = document.getElementById('app');

// create a root
const root = ReactDOM.createRoot(container);

//render app to root
root.render(<App />);
```

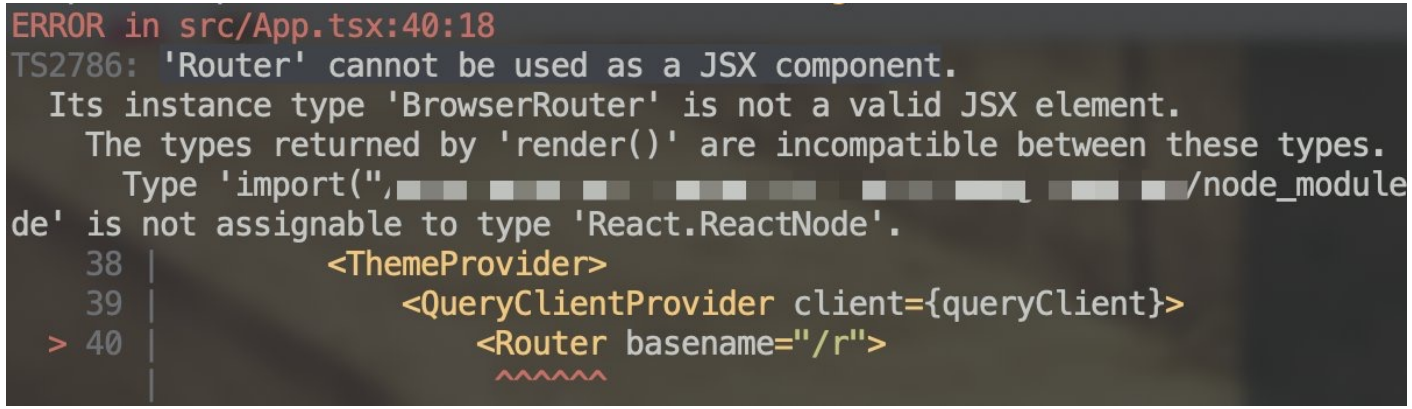
**renderToString** – continues to work with limited support, suggested to switch to **renderToPipeableStream** to unlock the new features.

# Potential dependency incompatibilities

**In most cases that shouldn't break your app, but some extra warnings might show up in terminal.**

- Some weird behavior is possible, especially when using **StrictMode**.

- Example packages, which can lead to additional warnings or bugs:
  - react-router v5
  - Enzyme and related adapters
  - Older @testing-library packages
  - Even @types/react packages!

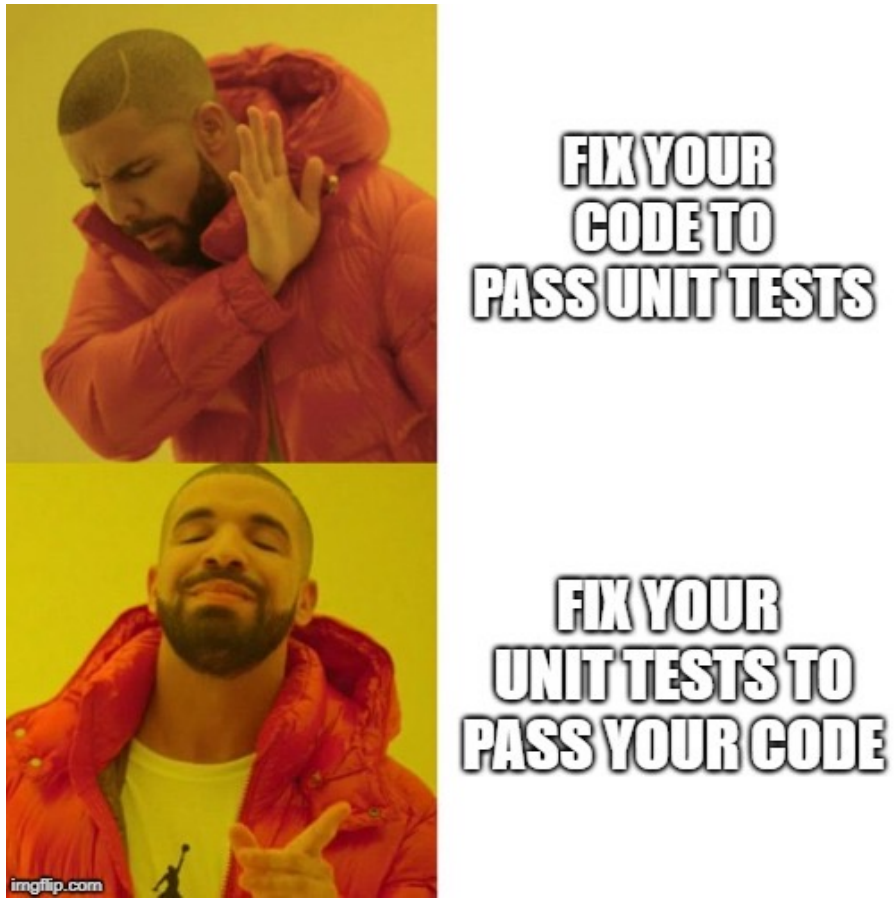



**p.s. When playing around with package versions, sometimes might be useful to remove your node_modules ☺**

# Unit testing

## In short: update to latest versions of React Testing Library

- If you are still using old packages like **Enzyme**, it's better to forget about those now.
  - Officially not supported already since v16.
- **@testing-library/react-hook** is deprecated, use original **@testing-library/react** instead.
- Some tests might be needed to be updates with extra **await**/**waitFor**/**act**.



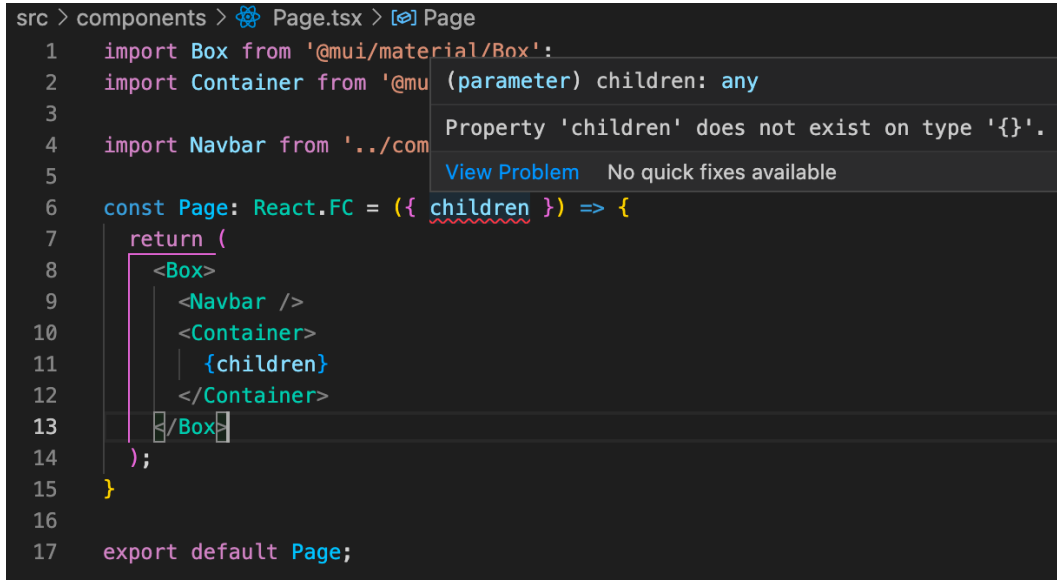
# Migration tips & tricks

**Some advices on how to avoid warnings and errors in terminal.**

- Upgrade not just **React**, but also **react-dom**, all related **@types/react*** packages, all **@testing-library/react*** and other possible React related packages.
- Upgrading **react-router** and **react-router-dom** is not mandatory but suggested.

- Change **ReactDOM.render** > **ReactDOM.createRoot**.
  - and import ReactDOM from 'react-dom/client' instead of 'react-dom'
- Use **React.StrictMode** for proper app migration.
- Suggested to use new APIs.

- Add children in component prop definition (removed in FC type).



# Personal experience

**In short: was easier than I expected!**

- Rewriting lots of unit tests from Enzyme to RTL or fixing some other tests...
- Weird bug in external package while using StrictMode.
- Introduced some flaky (in build pipeline only) Cypress tests (React rendering got faster).
- Most of the issues were already discussed in internet.

# Thank you!

## Any questions?

BALTICS