<epam>

React: Components

By Pavel Yukhnovich

July 2020

Agenda

- Core Concepts Recap
- Components
- Fragments
- Styling
- Advanced Styling
- Prop Types
- Advanced Type Checking
- Default Props
- Parent & Child
- Containers & Presentational Components
- Layout Component
- Error Boundaries
- Higher-Order Components
- Strict Mode
- Application Structure
- Components Wrapping Up
- Home Task Q&A

CORE CONCEPTS RECAP

Previously in the React Mentoring Program

- ES6+ features
- Declarative Programming Paradigm
- Functional Programming
- React is just a library
- JSX
- Virtual DOM
- Components

COMPONENTS

Components

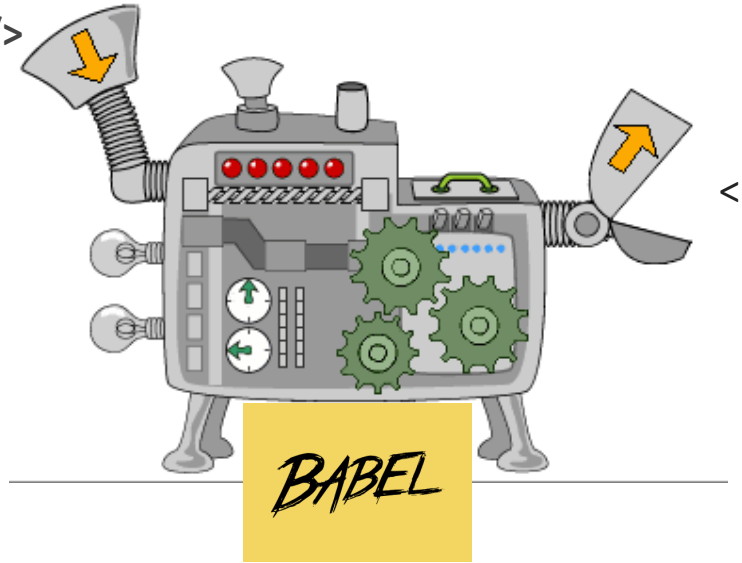
Components

Components are functions that accept arbitrary inputs (props) and return React elements describing what should appear on the screen

React

JSX

<Header/>



JS

<**h1**>Hello React</**h1**>;

Components via JSX

Function(al) component

```
import React from 'react'

let Header = () => <h1>Hello React</h1>

export default Header
```

```
import React from 'react'

function Header() {
  return <h1>Hello React!</h1>
}

export default Header
```

Class component (via Component)

```
import React from 'react'

class Header extends React.Component {
  render() {
    return <h1>Hello React</h1>
  }
}

export default Header
```

Class component (via PureComponent)

```
import React from 'react'

class Header extends React.PureComponent {
  render() {
    return <h1>Hello React</h1>
  }
}

export default Header
```

Components via JSX

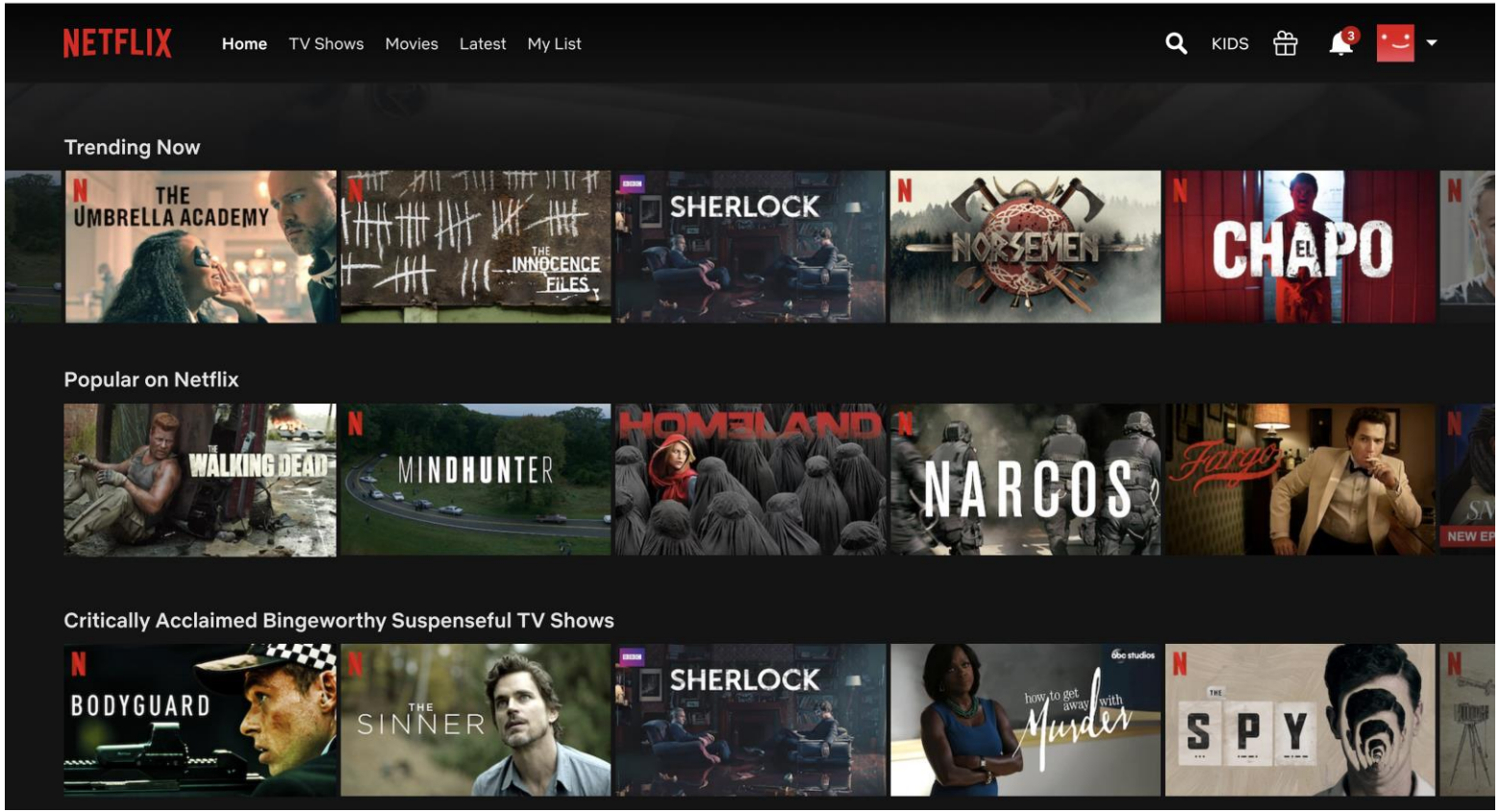
Function(al) component

```
import React from 'react'

let Header = () => <h1>Hello React</h1>

export default Header
```

```
import React from 'react'

function Header() {
  return <h1>Hello React!</h1>
}

export default Header
```

```
import React from 'react'

export default () => <h1>Hello React</h1>
```

Almost all web apps in … year

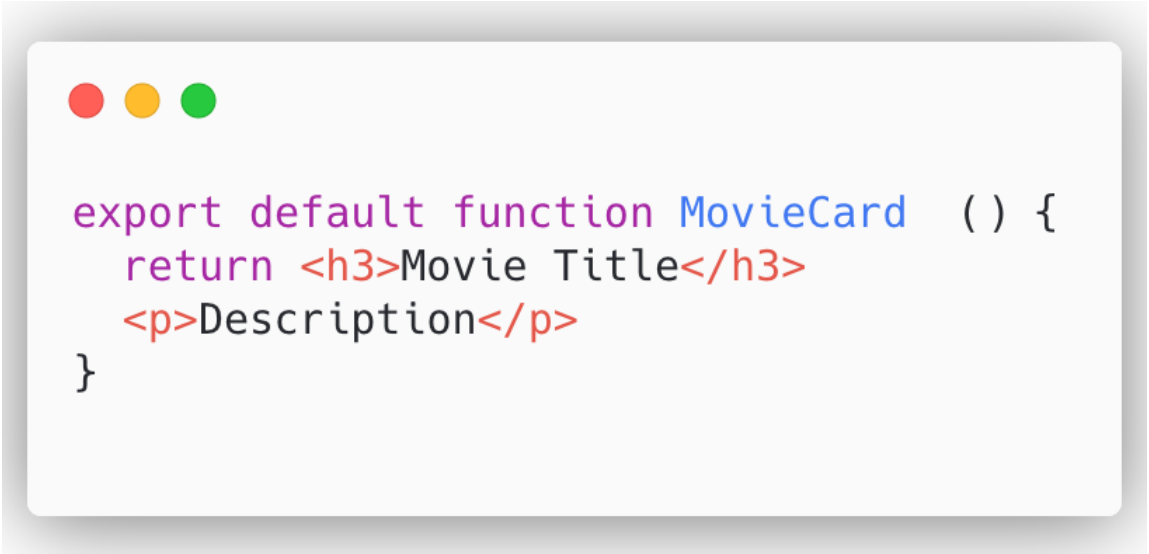


Components

```
import React from 'react'

export default function MovieCard() {
  return <h3>Movie Title</h3>
}
```

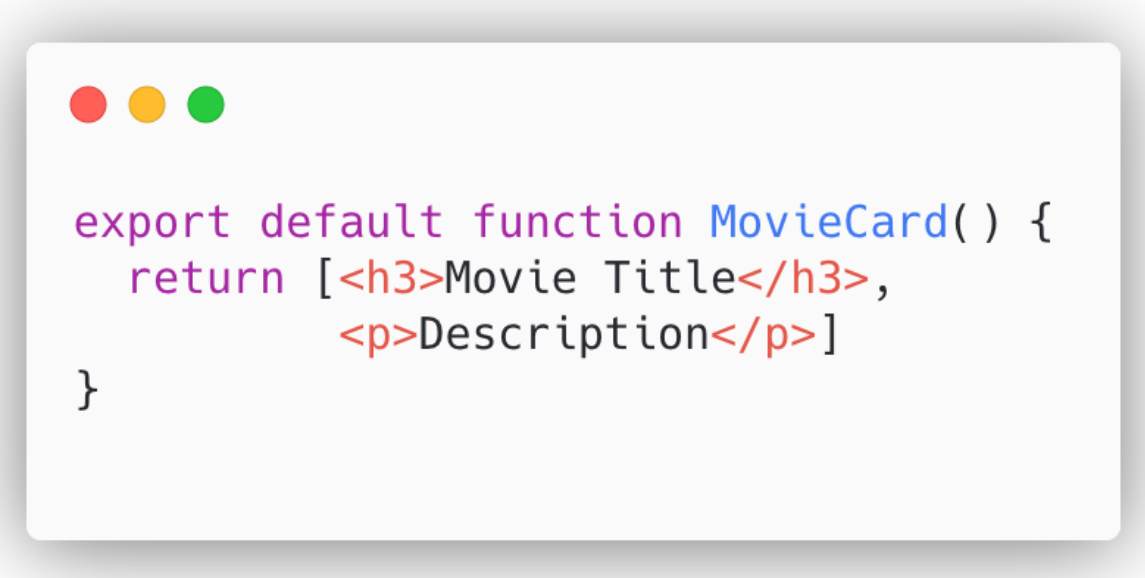
Components

Will not work



```
export default function MovieCard  () {
  return <h3>Movie Title</h3>
  <p>Description</p>
}
```

Components

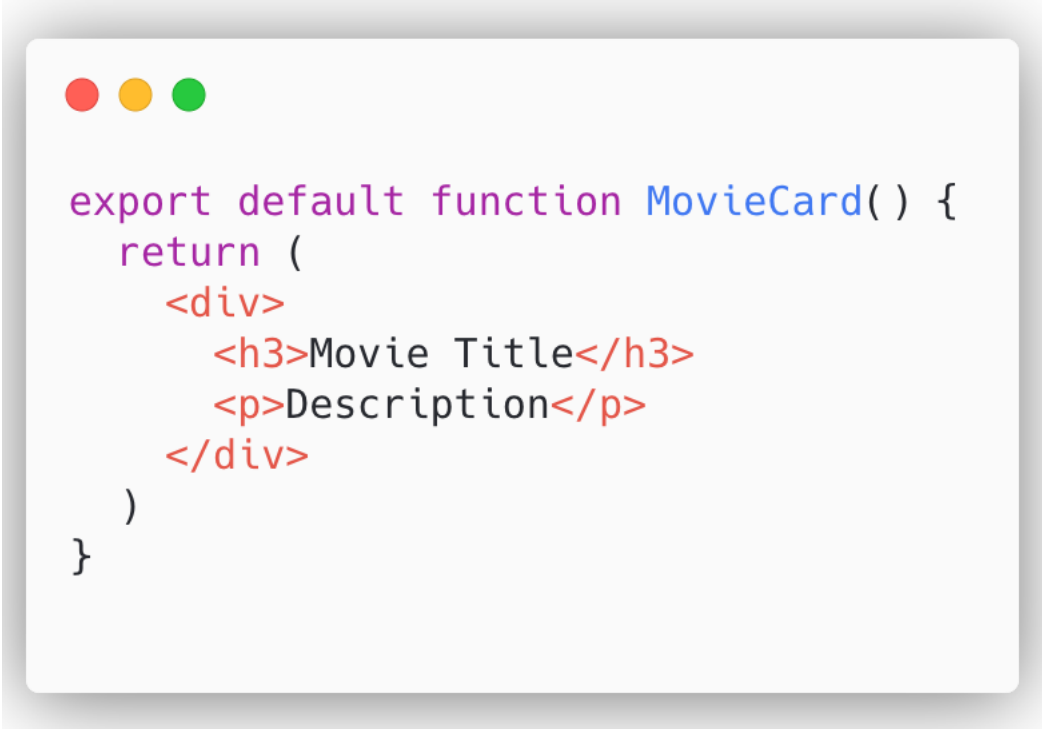


```
export default function MovieCard() {
  return [<h3>Movie Title</h3>,
          <p>Description</p>]
}
```

Components

```
export default function MovieCard  () {
  return [<h3 key='uniqueKey1'>Movie Title</h3>,
          <p key='uniqueKey2'>Description</p>]
}
```

Components



```
export default function MovieCard() {
  return (
    <div>
      <h3>Movie Title</h3>
      <p>Description</p>
    </div>
  )
}
```

Components

```
import React from 'react'

import MovieCard from './MovieCard'

export default function App() {
  return (
    <div>
      <MovieCard />
      <MovieCard />
      <MovieCard />
      <MovieCard />
    </div>
  )
}
```

Too many s ?

```jsx
import React from "react";

//import MovieCard from "./MovieCard";

export default function App() {
  return (
    <div>
      <div>
        <h3>Movie Title</h3>
        <p>Description</p>
      </div>
      <div>
        <h3>Movie Title</h3>
        <p>Description</p>
      </div>
      <div>
        <h3>Movie Title</h3>
        <p>Description</p>
      </div>
      <div>
        <h3>Movie Title</h3>
        <p>Description</p>
      </div>
    </div>
  );
}
```

hell

```
<div id="yet-another-1-wrapper">
    <div id="yet-another-2-wrapper">
        <div id="yet-another-3-wrapper">
            <div id="yet-another-4-wrapper">
                <div id="yet-another-5-wrapper">
                    <div id="yet-another-6-wrapper">
                        <div id="yet-another-7-wrapper">
                            <div id="yet-another-8-wrapper">
                                <div id="yet-another-9-wrapper">
                                    <div id="yet-another-10-wrapper">
                                        <div id="yet-another-11-wrapper">
                                            <div id="yet-another-12-wrapper">
                                                <div id="yet-another-13-wrapper">
                                                    <div id="yet-another-14-wrapper">
                                                        <div id="yet-another-15-wrapper">
                                                            <div id="yet-another-16-wrapper">
                                                                <h1>Some Heading</h1>
                                                                <p>some text</p>
                                                            </div>
                                                        </div>
                                                    </div>
                                                </div>
                                            </div>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

hell

A large DOM tree can slow down your page performance in multiple ways:

Network efficiency and load performance

A large DOM tree often includes many nodes that aren't visible when the user first loads the page, which unnecessarily increases data costs for your users and slows down load time.

Runtime performance

As users and scripts interact with your page, the browser must constantly recompute the position and styling of nodes. A large DOM tree in combination with complicated style rules can severely slow down rendering.

Memory performance

If your JavaScript uses general query selectors such as document.querySelectorAll('li'), you may be unknowingly storing references to a very large number of nodes, which can overwhelm the memory capabilities of your users' devices.

Broken Visual elements

Some CSS mechanisms like *Flexbox* and *CSS Grid* have a special parent-child relationships, and adding divs in the middle makes it hard to keep the desired layout

hell

Note from the real projects:
Lighthouse check might fail the pipeline

Lighthouse flags pages with DOM trees that:

- Have more than 1,500 nodes total.
- Have a depth greater than 32 nodes.
- Have a parent node with more than 60 child nodes.

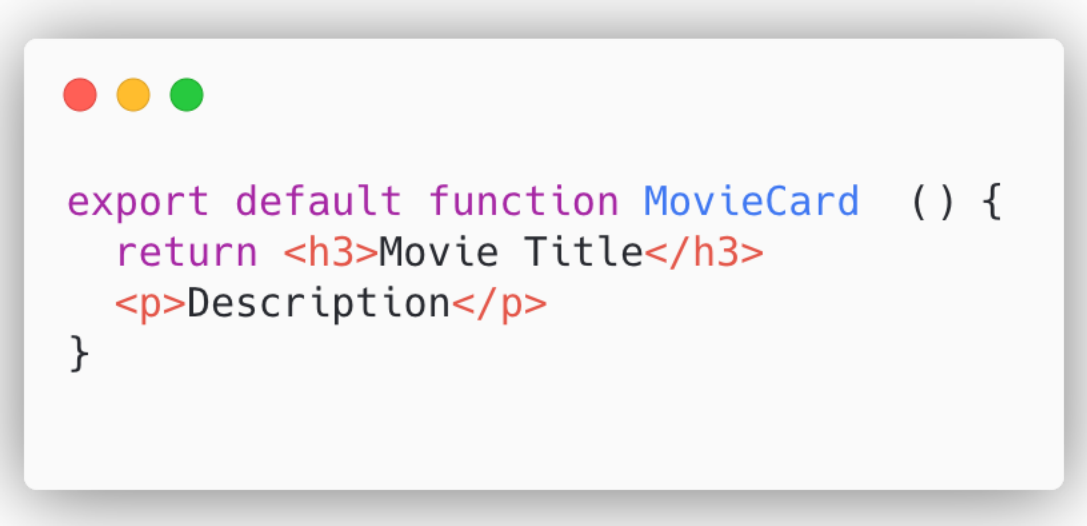
▲ **Avoid an excessive DOM size** — 3,071 elements

Browser engineers recommend pages contain fewer than ~1,500 DOM elements. The sweet spot is a tree depth < 32 elements and fewer than 60 children/parent element. A large DOM can increase memory usage, cause longer style calculations, and produce costly layout reflows. Learn more.

Statistic	Element	Value
Total DOM Elements		3,071
Maximum DOM Depth	`<input data-reftag="cm_cr_dp_mb_hlp_yes" class="a-button-input" type="submit" aria-labelledby="a-autoid-16-announce">`	22
Maximum Child Elements	`<div id="WrapperDivToHideBTF">`	69

Avoid unnecessary s

Will not work



```
export default function MovieCard  () {
  return <h3>Movie Title</h3>
  <p>Description</p>
}
```

FRAGMENTS

Avoid unnecessary s

```
export default function MovieCard  () {
  return <h3>Movie Title</h3>
  <p>Description</p>
}
```

```
▶ SyntaxError: /src/MovieCard.js: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? (5:2)
```

React.Fragment

```
import React from 'react'

export default function MovieCard() {
  return (
    <>
      <h3>Movie Title</h3>
      <p>Description</p>
    </>
  )
}
```

```
▶ SyntaxError: /src/MovieCard.js: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? (5:2)
```

React.Fragment

```jsx
import React from 'react'

export default function MovieCard() {
  return (
    <>
      <h3>Movie Title</h3>
      <p>Description</p>
    </>
  )
}
```

```jsx
import React, { Fragment } from 'react'

export default function MovieCard() {
  return (
    <Fragment>
      <h3>Movie Title</h3>
      <p>Description</p>
    </Fragment>
  )
}
```

```jsx
import React from "react";

export default function MovieCard() {
  return (
    <React.Fragment>
      <h3>Movie Title</h3>
      <p>Description</p>
    </React.Fragment>
  );
}
```

React.Fragment

```jsx
import React from 'react'

export default function MovieCard() {
  return (
    <>
      <h3>Movie Title</h3>
      <p>Description</p>
    </>
  )
}
```

```jsx
import React, { Fragment } from 'react'

export default function MovieCard() {
  return (
    <Fragment>
      <h3>Movie Title</h3>
      <p>Description</p>
    </Fragment>
  )
}
```

```jsx
import React from "react";

export default function MovieCard() {
  return (
    <React.Fragment>
      <h3>Movie Title</h3>
      <p>Description</p>
    </React.Fragment>
  );
}
```

React.Fragment

```
import React from 'react'

import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <MovieCard />
      <MovieCard />
      <MovieCard />
      <MovieCard />
    </>
  )
}
```

```
... ▼<div id="root"> == $0
      <h3>Movie Title</h3>
      <p>Description</p>
      <h3>Movie Title</h3>
      <p>Description</p>
      <h3>Movie Title</h3>
      <p>Description</p>
      <h3>Movie Title</h3>
      <p>Description</p>
    </div>
```

- Virtual Dom won't have redundant jobs
- Added features not possible before with JSX
- Better semantic jsx markup.

STYLING

Styling: things to consider

- Global name spacing
- Dependencies
- Reusability
- Scalability
- Dead-code Elimination

Styling: Inline Styles

- Might be ok for small apps
- Styles are readable

```
import React from 'react'

const cardTitleStyles = {
  margin: 5,
  textDecoration: 'underline',
}

const MovieCard = ({ title }) => (
  <div style={{ marginTop: '15px' }}>
    <p style={cardTitleStyles}>{title}</p>
  </div>
)

export default MovieCard
```

- Bad for performance
- Confusing camelCase CSS
- Bad for huge applications
- Media queries support
- Animations support
- Pseudo-classes reference
- Pseudo-elements reference

- Dependencies: **None**
- Difficulty: **Easy**
- Approach: **Worst**

Styling: Regular CSS

- Good for large apps
- Reusable styles
- Separated styles

footer.css

```
.footerMessage {
  margin: 40px;
  border: 5px dotted red;
  border-radius: 5px;
 }
```

- Bad for huge apps

```
import React from 'react'
import './footer.css'

let Footer = () => <p className="footerMessage">This is footer</p>

export default Footer
```

- Dependencies: **None**
- Difficulty: **Easy**
- Approach: **So-so**

Styling: Sass/ SCSS

- Ok for large apps
- Reusable styles
- Separated styles

footer.scss

```scss
.footerMessage {
  margin: 40px;
  border: 5px dotted red;
  border-radius: 5px;
 }
```

- *Separated styles
- Name collisions

```jsx
import React from 'react'
import './footer.scss'

let Footer = () => <p className="footerMessage">This is footer</p>

export default Footer
```

- Dependencies: **node-sass**
- Difficulty: **Easy**
- Approach: **Best**

Styling: CSS Modules

- Great for large apps
- No scope issues
- Reusable

- Overkill for small apps
- Might be difficult to configure

footer.module.scss

```scss
.content {
  font-size: 15px;
  text-align: center;
 }
```

```jsx
import React from "react";
import styles from './footer.module.scss';

const Paragraph = () => <p className={styles.content}>This is a paragraph with CSS module</p>

export default Paragraph;
```

- Dependencies: **css-loader**
- Difficulty: **Tough** (Uses Loader Configuration)
- Approach: **Best**

* Styling with Template literals (Template strings)

- Time saver
- Zero dependencies
- Robust

- Not enough for a large app

```
<li className={`movieCard watched`}>Interstellar</li>

<li className={`movieCard ${props.isWatched ? 'watched' : ''}`}>Interstellar</li>

<li className={'movieCard ' + (props.isWatched ? 'watched' : '') }>Interstellar</li>
```

- Dependencies: **none**
- Difficulty: **Easy**
- Approach: **ok**

ADVANCED STYLING

CSS in JS (JSS)


```
var font = {
  fontSize: 12,
  lineHeight: 1,
}

export default {
  button: {
    extend: font,
    border: 'none',
    margin: [5, 10],
    transition: ['background', 'color', 'font-size'],
    transitionDuration: 300,
    background: {
      color: 'white',
      image: 'url("/some/url/image.png")',
      repeat: 'no-repeat',
      position: 'contain',
    },
    '&:before': {
      content: '"icon"',
    },
  },
  redButton: {
    extend: 'button',
    background: 'linear-gradient(to right, red 0%, green 100%)',
    fallbacks: {
      background: 'red',
    },
    '&:hover': {
      border: [[1, 'solid', 'blue']],
      boxShadow: [
        [0, 0, 0, 10, 'blue'],
        [0, 0, 0, 15, 'green'],
      ],
      '& span': {
        color: 'red',
      },
    },
  },
  '@media (min-width: 1024px)': {
    button: {
      fontSize: 16,
    },
  },
}
```

```
.button-0-1-7 {
  border: none;
  margin: 5px 10px;
  font-size: 12px;
  background: white contain no-repeat;
  transition: background color font-size;
  line-height: 1;
  background-image: url("/some/url/image.png");
  transition-duration: 300ms;
}
.button-0-1-7:before {
  content: "icon";
}
.redButton-0-1-8 {
  background: red;
  border: none;
  margin: 5px 10px;
  font-size: 12px;
  background: linear-gradient(to right, red 0%, green 100%);
  transition: background color font-size;
  line-height: 1;
  transition-duration: 300ms;
}
.redButton-0-1-8:before {
  content: "icon";
}
.redButton-0-1-8:hover {
  border: 1px solid blue;
  box-shadow: 0 0 0 10px blue, 0 0 0 15px green;
}
.redButton-0-1-8:hover span {
  color: red;
}
@media (min-width: 1024px) {
  .button-0-1-7 {
    font-size: 16px;
  }
}
```

CSS in JS (JSS)


```jsx
import React from 'react'
import {render} from 'react-dom'
import {createUseStyles} from 'react-jss'

// Create your Styles. Remember, since React-JSS uses the default preset,
// most plugins are available without further configuration needed.
const useStyles = createUseStyles({
  myButton: {
    color: 'green',
    margin: {
      // jss-plugin-expand gives more readable syntax
      top: 5, // jss-plugin-default-unit makes this 5px
      right: 0,
      bottom: 0,
      left: '1rem'
    },
    '& span': {
      // jss-plugin-nested applies this to a child span
      fontWeight: 'bold' // jss-plugin-camel-case turns this into 'font-weight'
    }
  },
  myLabel: {
    fontStyle: 'italic'
  }
})

// Define the component using these styles and pass it the 'classes' prop.
// Use this to assign scoped class names.
const Button = ({children}) => {
  const classes = useStyles()
  return (
    <button className={classes.myButton}>
      <span className={classes.myLabel}>{children}</span>
    </button>
  )
}

const App = () => <Button>Submit</Button>

render(<App />, document.getElementById('root'))
```

CSS in JS (JSS)

```jsx
import React from 'react'
import {render} from 'react-dom'
import {createUseStyles} from 'react-jss'

// Create your Styles. Remember, since React-JSS uses the default preset,
// most plugins are available without further configuration needed.
const useStyles = createUseStyles({
  myButton: {
    color: 'green',
    margin: {
      // jss-plugin-expand gives more readable syntax
      top: 5, // jss-plugin-default-unit makes this 5px
      right: 0,
      bottom: 0,
      left: '1rem'
    },
    '& span': {
      // jss-plugin-nested applies this to a child span
      fontWeight: 'bold' // jss-plugin-camel-case turns this into 'font-weight'
    }
  },
  myLabel: {
    fontStyle: 'italic'
  }
})

// Define the component using these styles and pass it the 'classes' prop.
// Use this to assign scoped class names.
const Button = ({children}) => {
  const classes = useStyles()
  return (
    <button className={classes.myButton}>
      <span className={classes.myLabel}>{children}</span>
    </button>
  )
}

const App = () => <Button>Submit</Button>

render(<App />, document.getElementById('root'))
```

```html
<div id="root">
  <button class="Button-myButton-1-25">
    <span class="Button-myLabel-1-26">
      Submit
    </span>
  </button>
</div>
```

```css
.Button-myButton-1-25 {
  color: green;
  margin: 5px 0 0 1rem;
}
.Button-myButton-1-25 span {
  font-weight: bold;
}
.Button-myLabel-1-26 {
  font-style: italic;
}
```

CSS in JS

1.Real CSS.
2.JSS generates actual CSS, not Inline Styles. It supports every existing CSS feature. CSS rules are created once and reused across the elements using its class name in contrary to Inline Styles. Also, when DOM elements get updated, previously created CSS rules are applied.
3.Collision-free selectors.
4.JSS generates unique class names by default. It allows avoiding the typical CSS problem, where everything is global by default. It completely removes the need for naming conventions.
5.Code reuse.
6.Using JavaScript as a host language gives us an opportunity to reuse CSS rules in a way that is not possible with regular CSS. You can leverage JavaScript modules, variables, functions, math operations and more. If done right, it can still be fully declarative.
7.Ease of removal and modification.
8.Explicit usage of CSS rules allows you to track them down to the consumer and decide if it can be safely removed or modified.
9.Dynamic styles.
10.Using JavaScript functions and Observables makes it possible to dynamically generate styles in the browser, giving you an opportunity to access your application state, browser APIs or remote data for styling. You can not only define styles once but also update them at any point in time in an efficient way.
11.User-controlled animations.
12.JSS handles CSS updates so efficiently that you can create complex animations with it. Using function values, Observables and combining them with CSS transitions will give you maximum performance for user-controlled animations. For predefined animations, it is still better to use @keyframes and transitions, because they will unblock the JavaScript thread completely.
13.Critical CSS.
14.To optimize time to first paint, you can use server-side rendering and extract critical CSS. You can couple the rendering of CSS with the rendering of HTML so that no unused CSS gets generated. It will result in a minimal critical CSS extracted during server-side rendering and allow you to inline it.
15.Plugins.
16.JSS core implements a plugin-based architecture. It allows you to create custom plugins which can implement custom syntax or other powerful abilities. JSS has many official plugins, which can be installed individually or using a default preset. A good example of a community plugin is jss-rtl.
17.Expressive syntax.
18.Thanks to various plugins, JSS allows you to have nesting, global selectors, composition with existing global class names. E.g. jss-plugin-expand allows you to express properties like box-shadow even in a more readable way than it is possible with CSS. You can also use template strings if you want to copy-paste styles directly from the browser dev tools.
19.Full isolation.
20.Another useful plugin example is jss-plugin-isolate, which allows you to isolate your elements from global cascading rules fully and potentially overwriting unwanted properties. Especially useful when creating a widget that is supposed to render inside of a third-party document …

CSS in JS

The React-JSS package provides some additional features:

- **Dynamic Theming** - allows context based theme propagation and runtime updates.
- **Critical CSS extraction** - only CSS from rendered components gets extracted.
- **Lazy evaluation** - Style Sheets get created when a component gets mounted.
- The **static part** of a Style Sheet gets shared between all elements.
- Function values and rules are updated **automatically** with props as an argument.

- Overkill for small apps
- Dependencies
- Vendor lock in
- Issues with hooks support
- Issues with TypeScript support

Styling: Styled Components


```jsx
import React from 'react'

import styled from 'styled-components'

// Create a <Title> react component that renders an <h1> which is
// centered, palevioletred and sized at 1.5em
const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: palevioletred;
`

// Create a <Wrapper> react component that renders a <section> with
// some padding and a papayawhip background
const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`

// Use them like any other React component – except they're styled!
let StyledBlock = () => (
  <Wrapper>
    <Title>Hello World, this is my first styled component!</Title>
  </Wrapper>
)

export default StyledBlock
```

Styling: Styled Components

•**Automatic critical CSS**: styled-components keeps track of which components are rendered on a page and injects their styles and nothing else, fully automatically. Combined with code splitting, this means your users load the least amount of code necessary.

•**No class name bugs**: styled-components generates unique class names for your styles. You never have to worry about duplication, overlap or misspellings.

•**Easier deletion of CSS**: it can be hard to know whether a class name is used somewhere in your codebase. styled-components makes it obvious, as every bit of styling is tied to a specific component. If the component is unused (which tooling can detect) and gets deleted, all its styles get deleted with it.

•**Simple dynamic styling**: adapting the styling of a component based on its props or a global theme is simple and intuitive without having to manually manage dozens of classes.

•**Painless maintenance**: you never have to hunt across different files to find the styling affecting your component, so maintenance is a piece of cake no matter how big your codebase is.

•**Automatic vendor prefixing**: write your CSS to the current standard and let styled-components handle the rest.
You get all of these benefits while still writing the CSS you know and love, just bound to individual components.

Styling: Styled Components

Ecosystem:

- Components
- Grid Systems
- Helpers
- Testing
- Boilerplates
- Real Apps (Patreon, Vimeo, Target, InVision, Vogue, etc.)

Styling: Styled Components

Ecosystem:

- Components
- Grid Systems
- Helpers
- Testing
- Boilerplates
- Real Apps (Patreon, Vimeo, Target, InVision, Vogue, etc.)

- Overkill for small apps
- Dependencies
- Styles and JS are mixed
- Vendor lock in
- *Performance issues

Styling: React Component Libraries

- Time savers
- Already known UI

Danger!

reactstrap

```
import React from 'react'
import { Button } from 'reactstrap'

export default (props) => {
  return <Button color="danger">Danger!</Button>
}
```

- Hard to modify
- Dependencies
- Collisions between libraries
- *Possible performance issues

- Dependencies: **depends**
- Difficulty: **Easy**
- Approach: **ok**

- Material UI
- ANT Design
- ReactStrap etc

PROP TYPES

Why Types

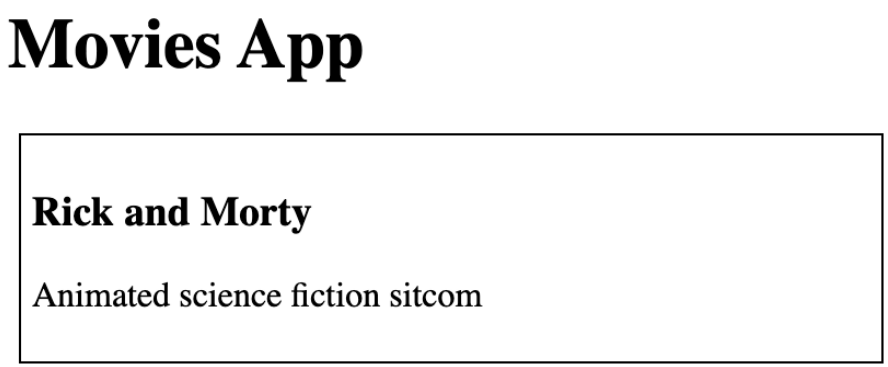
```
import React from 'react'

export default () => <h1>Movies App</h1>
```

```
import React from 'react'

import './styles.css'

export default function MovieCard(props) {
  return (
    <div className="MovieCard">
      <h3>{props.title}</h3>
      <p>{props.description}</p>
    </div>
  )
}
```

```
import React from 'react'

import Header from './Header'
import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <Header />
      <MovieCard
        title="Rick and Morty"
        description="Animated science fiction sitcom"
      />
    </>
  )
}
```

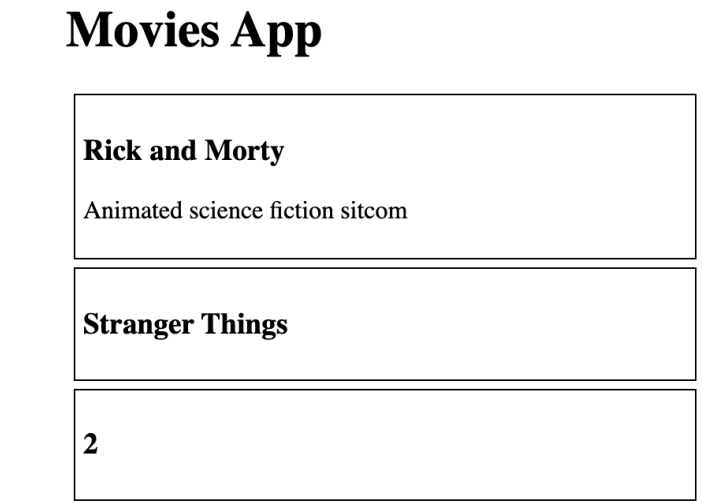
Why Types

```jsx
import React from 'react'

import Header from './Header'
import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <Header />
      <MovieCard
        title="Rick and Morty"
        description="Animated science fiction sitcom"
      />
    </>
  )
}
```



Why Types

```jsx
import React from 'react'

import Header from './Header'
import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <Header />
      <MovieCard
        title="Rick and Morty"
        description="Animated science fiction sitcom"
      />
      <MovieCard title="Stranger Things" />
      <MovieCard title={2} />
    </>
  )
}
```



Prop Types

```
import React from 'react'

import PropTypes from 'prop-types'

import './styles.css'

export default function MovieCard(props) {
  return (
    <div className="MovieCard">
      <h3>{props.title}</h3>
      <p>{props.description}</p>
    </div>
  )
}

MovieCard.propTypes = {
  title: PropTypes.string.isRequired,
  description: PropTypes.string.isRequired,
}
```

```
npm install --save prop-types
```

Movies App

Rick and Morty

Animated science fiction sitcom

Stranger Things

2

```
▶ Warning: Failed prop type: The prop `description` is marked as required in `MovieCard`, but its value is `undefined`.
▶ Warning: Failed prop type: Invalid prop `title` of type `number` supplied to `MovieCard`, expected `string`.
```

Prop Types

- Great library to type-check your application
- Helps with understanding issues at the early stage
- Checks JS types, React elements, required props, and so on
- Might be considered as a documentation tool

- Will not help in the production mode
- Flow or TypeScript provide more features

```
MyComponent.propTypes = {
  // You can declare that a prop is a specific JS primitive. By default, these
  // are all optional.
  optionalArray: PropTypes.array,
  optionalBool: PropTypes.bool,
  optionalFunc: PropTypes.func,
  optionalNumber: PropTypes.number,
  optionalObject: PropTypes.object,
  optionalString: PropTypes.string,
  optionalSymbol: PropTypes.symbol,

  // Anything that can be rendered: numbers, strings, elements or an array
  // (or fragment) containing these types.
  optionalNode: PropTypes.node,

  // A React element (ie. <MyComponent />).
  optionalElement: PropTypes.element,

  // A React element type (ie. MyComponent).
  optionalElementType: PropTypes.elementType,

  // You can also declare that a prop is an instance of a class. This uses
  // JS's instanceof operator.
  optionalMessage: PropTypes.instanceOf(Message),

  // You can ensure that your prop is limited to specific values by treating
  // it as an enum.
  optionalEnum: PropTypes.oneOf(['News', 'Photos']),

  // An object that could be one of many types
  optionalUnion: PropTypes.oneOfType([
    PropTypes.string,
    PropTypes.number,
    PropTypes.instanceOf(Message)
  ]),

  // An array of a certain type
  optionalArrayOf: PropTypes.arrayOf(PropTypes.number),

  // An object with property values of a certain type
  optionalObjectOf: PropTypes.objectOf(PropTypes.number),

  // You can chain any of the above with `isRequired` to make sure a warning
  // is shown if the prop isn't provided.

  // An object taking on a particular shape
  optionalObjectWithShape: PropTypes.shape({
    optionalProperty: PropTypes.string,
    requiredProperty: PropTypes.number.isRequired
  }),
```

ADVANCED TYPE CHECKING

Advanced Type Checking

TypeScript 

Flow 

	TypeScript	Flow
Editor and IDE support	Widespread	Little to no support
Questions posted on Stack Overflow	100000+	600+
Framework support	Many, including Express, Vue, React, Angular, etc.	React only
Library support	Many	Few to none (that we know of)
Autocomplete	Available in IDEs and text editors	None
Compiler error detection	Available in IDEs and text editors	None
Syntax	Comprehensive type checking, includes both static and dynamic type annotations	Comprehensive type checking, includes both static and dynamic type annotations
Generics	Supported	Supported
Support in existing projects	TypeScript package can be added to support TypeScript	Add support with Babel

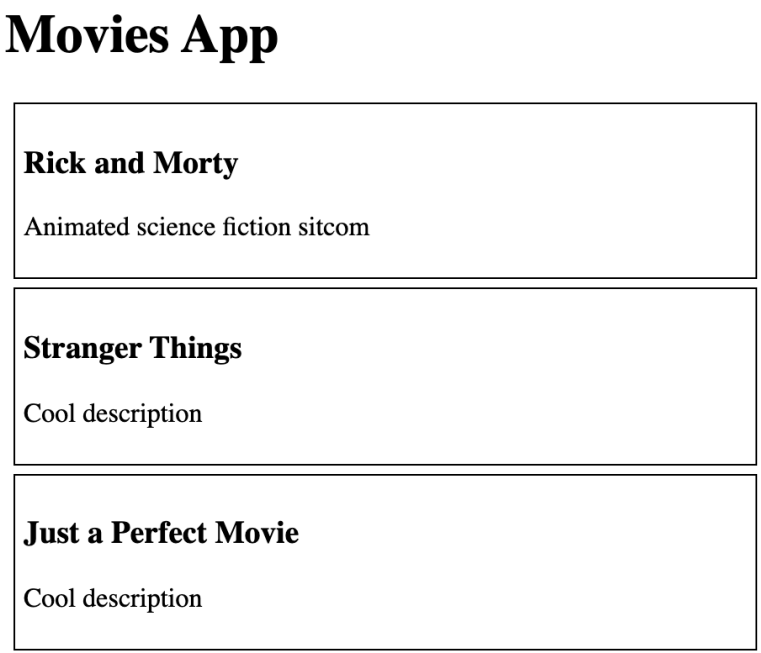
DEFAULT PROPS

Default Props

```jsx
import React from 'react'

import Header from './Header'
import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <Header />
      <MovieCard
        title="Rick and Morty"
        description="Animated science fiction sitcom"
      />
      <MovieCard title="Stranger Things" />
      <MovieCard />
    </>
  )
}
```

Default Props

```
import React from 'react'

import './styles.css'

export default function MovieCard(props) {
  return (
    <div className="MovieCard">
      <h3>{props.title}</h3>
      <p>{props.description}</p>
    </div>
  )
}

MovieCard.defaultProps = {
  title: 'Just a Perfect Movie',
  description: 'Cool description',
}
```



PARENT & CHILD

Parent - Child

```jsx
import React from 'react'

import Header from './Header'
import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <Header />
      <MovieCard
        title="Rick and Morty"
        description="Animated science fiction sitcom"
      />
      <MovieCard title="Stranger Things" />
      <MovieCard />
    </>
  )
}
```

Movies App

Rick and Morty

Animated science fiction sitcom

Stranger Things

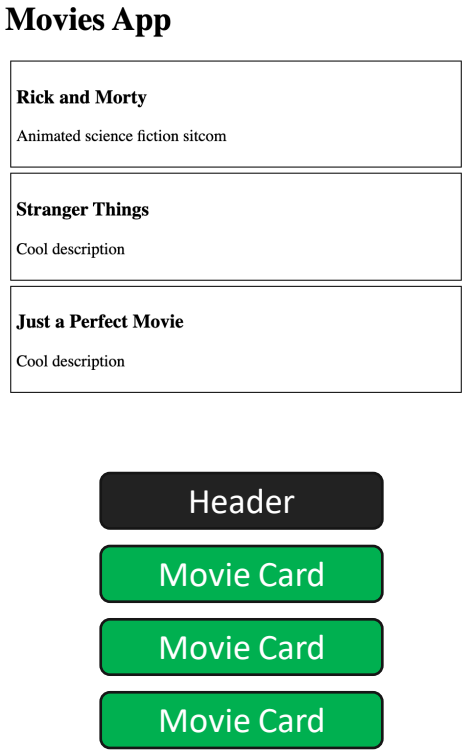
Cool description

Just a Perfect Movie

Cool description

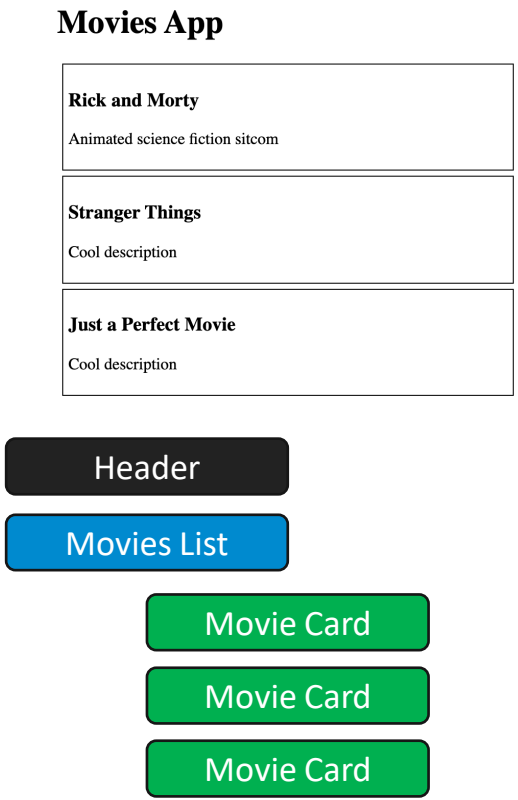
Parent - Child

```jsx
import React from 'react'

import Header from './Header'
import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <Header />
      <MovieCard
        title="Rick and Morty"
        description="Animated science fiction sitcom"
      />
      <MovieCard title="Stranger Things" />
      <MovieCard />
    </>
  )
}
```



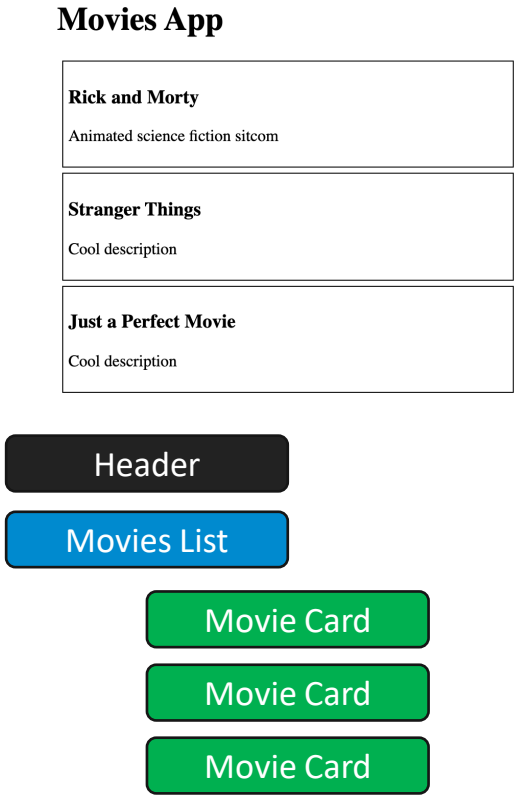
Parent - Child

```jsx
import React from 'react'

import Header from './Header'
import MovieCard from './MovieCard'

export default function App() {
  return (
    <>
      <Header />
      <MovieCard
        title="Rick and Morty"
        description="Animated science fiction sitcom"
      />
      <MovieCard title="Stranger Things" />
      <MovieCard />
    </>
  )
}
```



Parent - Child

```
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'

export default function App() {
  return (
    <>
      <Header />
      <MoviesList />
    </>
  )
}
```



Parent - Child

```jsx
import React from 'react'

import MovieCard from './MovieCard'

//lets assume we have this data from API
let movies = [
  {
    title: 'Rick and Morty',
    description: 'Animated science fiction sitcom',
    id: 'r43d2',
  },
  {
    title: 'Stranger Things',
    description: 'American science fiction series',
    id: 'f43ds',
  },
  {
    title: 'Interstellar',
    description: 'Epic science fiction movie',
    id: 'd2s24',
  },
]

export default function MoviesList() {
  return (
    <>
      {movies.map((movie) => (
        <MovieCard
          title={movie.title}
          description={movie.description}
          key={movie.id}
        />
      ))}
    </>
  )
}
```

Movies List

Movie Card

Movie Card

Movie Card

Movies App

Rick and Morty

Animated science fiction sitcom

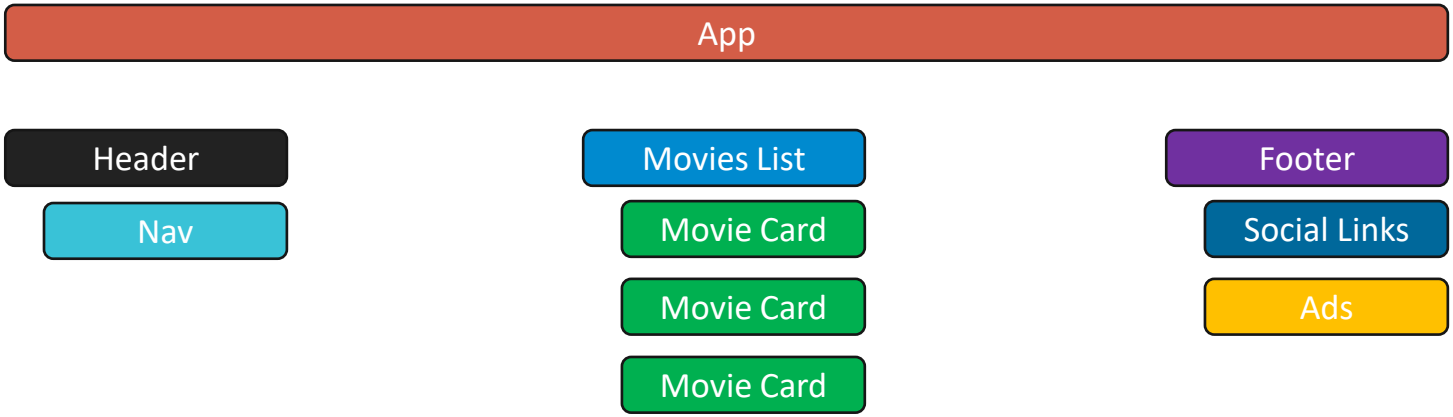
Stranger Things

American science fiction series

Interstellar

Epic science fiction movie

Parent - Child



Parent - Child

```jsx
import React from 'react'

import MovieCard from './MovieCard'

//lets assume we have this data from API
let movies = [
  {
    title: 'Rick and Morty',
    description: 'Animated science fiction sitcom',
    id: 'r43d2',
  },
  {
    title: 'Stranger Things',
    description: 'American science fiction series',
    id: 'f43ds',
  },
  {
    title: 'Interstellar',
    description: 'Epic science fiction movie',
    id: 'd2s24',
  },
]

export default function MoviesList() {
  return (
    <>
      {movies.map((movie) => (
        <MovieCard
          title={movie.title}
          description={movie.description}
          key={movie.id}
        />
      ))}
    </>
  )
}
```

```jsx
import React from 'react'

import './styles.css'

export default function MovieCard({ title, description }) {
  return (
    <div className="MovieCard">
      <h3>{title}</h3>
      <p>{description}</p>
    </div>
  )
}
```

Movies App

Rick and Morty

Animated science fiction sitcom

Stranger Things

American science fiction series

Interstellar

Epic science fiction movie

SMART & DUMB COMPONENTS
CONTAINERS & PRESENTATIONAL COMPONENTS

Smart & Dumb Components / Containers & Presentational Components

Container / Smart

- Knows how everything works
- Just a container for other components
- Provides application data
- Performs data fetching
- No idea about styles
- Might have side effects
- Might be implemented with impure functions

Presentational / Dumb

- Knows how to look
- Knows how to display received data
- Receives and operates only with props
- Operates with styles
- Works via pure functions
- Shouldn’t produce side effects

Smart & Dumb Components

Benefits of This Approach

- **Better separation of concerns.**

You understand your app and your UI better by writing components this way.

- **Better reusability.**

You can use the same presentational component with completely different state sources, and turn those into separate container components that can be further reused.

- **Presentational components are essentially your app's "palette".**

You can put them on a single page and let the designer tweak all their variations without touching the app's logic.
You can run screenshot regression tests on that page.

- **Layout components introduction.**

This forces you to extract "layout components" such as Navigation, Page, ContextMenu and use this.props.children instead of duplicating the same markup and layout in several container components.

LAYOUT COMPONENT

Layout Component

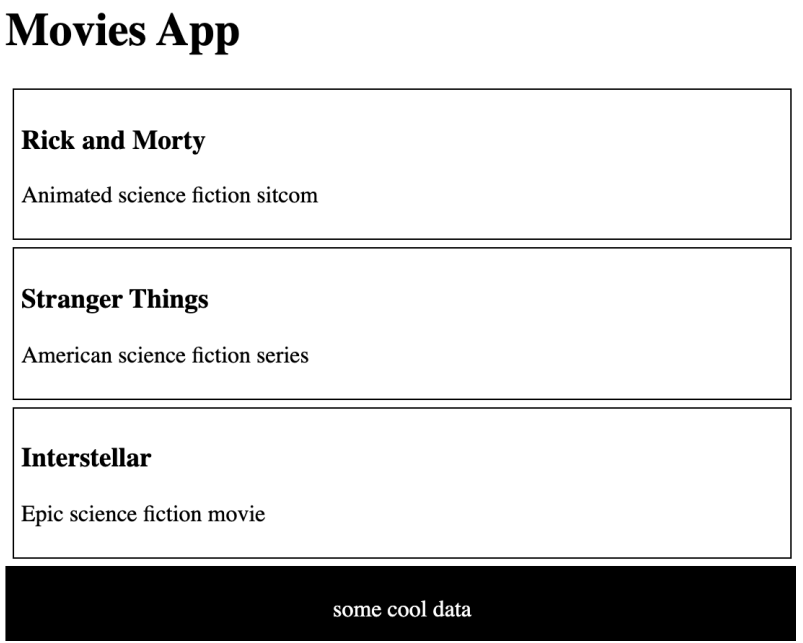
```jsx
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'
import Footer from './Footer'

export default function App() {
  return (
    <>
      <Header />
      <MoviesList />
      <Footer />
    </>
  )
}
```

```jsx
import React from 'react'
import './styles.css'

const Footer = (props) => <div className="Footer">{props.children}</div>

export default Footer
```

Layout Component

```jsx
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'
import Footer from './Footer'

export default function App() {
  return (
    <>
      <Header />
      <MoviesList />
      <Footer>
        <p> some cool data </p>
      </Footer>
    </>
  )
}
```

```jsx
import React from 'react'
import './styles.css'

const Footer = (props) => <div className="Footer">{props.children}</div>

export default Footer
```



Layout Component

Benefits of This Approach

- Better separation of concerns.
- Better reusability.
- Easy way to organize UI elements.
- No cross-team dependencies

ERROR BOUNDARIES

Error Boundary

```jsx
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'
import Footer from './Footer'

export default function App() {
  return (
    <>
      <Header />
      <MoviesList />
      <Footer />
    </>
  )
}
```

Error Boundary

```
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'
import Footer from './Footer'

export default function App() {
  return (
    <>
      <Header />
      <MoviesList />
      <Footer />
    </>
  )
}
```

In the past, JavaScript errors inside components used to corrupt React's internal state and cause it to emit critical errors on next renders.

React **did not provide** any way to handle them gracefully in components, and **could not recover from them**.

Error Boundary

```jsx
import React from 'react'

import './styles.css'

export default function MovieCard(props) {
  return (
    <div className="MovieCard">
      <h3>{props.title}</h3>
      <p>{props.description}</p>
    </div>
  )
}
```

```jsx
import React from 'react'

import MovieCard from './MovieCard'

//lets assume we have this data from API
let movies = [
  {
    title: 'Rick and Morty',
    description: 'Animated science fiction sitcom',
    id: 'r43d2',
  },
  {
    title: 'Stranger Things',
    description: 'American science fiction series',
    id: 'f43ds',
  },
  {
    title: 'Interstellar',
    description: 'Epic science fiction movie',
    id: 'd2s24',
  },
]

export default function MoviesList() {
  return (
    <>
      {movies.map((movie) => (
        <MovieCard
          title={movie.title}
          description={movie.description}
          key={movie.id}
        />
      ))}
    </>
  )
}
```

```jsx
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'
import Footer from './Footer'

export default function App() {
  return (
    <>
      <Header />
      <MoviesList />
      <Footer />
    </>
  )
}
```

Error Boundary

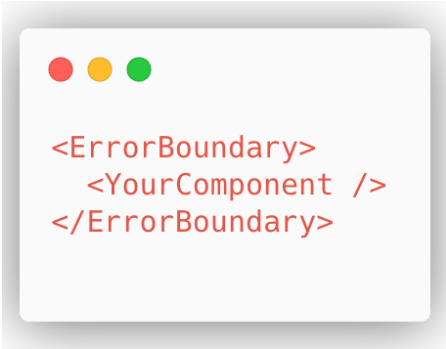
TypeError: this.props is not a function

```
App.render
webpack:///./components/App.js?:30:12

App.eval
webpack:///./~/react-hot-loader/~/react-hot-api/modules/makeAssimilatePrototype.js?:15:37

ReactCompositeComponentMixin._renderValidatedComponentWithoutOwnerOrContex
webpack:///./~/react/lib/ReactCompositeComponent.js?:789:34

ReactCompositeComponentMixin._renderValidatedComponent
webpack:///./~/react/lib/ReactCompositeComponent.js?:816:14

wrapper
webpack:///./~/react/lib/ReactPerf.js?:70:21

ReactCompositeComponentMixin.mountComponent
webpack:///./~/react/lib/ReactCompositeComponent.js?:237:30

wrapper
webpack:///./~/react/lib/ReactPerf.js?:70:21
```

Error Boundary

Error boundaries are components that

- catch JavaScript errors anywhere in their child component tree
- log those errors
- display a fallback UI instead of the component tree that crashed.



**A class component becomes an error boundary if it defines new lifecycle methods: componentDidCatch(error, info) or static getDerivedStateFromError()*

Error Boundary

Below are the cases in which error boundaries don't work

- Inside Event handlers
- Asynchronous code using setTimeout or requestAnimationFrame callbacks
- During Server side rendering
- When errors thrown in the error boundary code itself

Error Boundary

```jsx
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'
import Footer from './Footer'

export default function App() {
  return (
    <>
      <Header />
      <MoviesList />
      <Footer />
    </>
  )
}
```

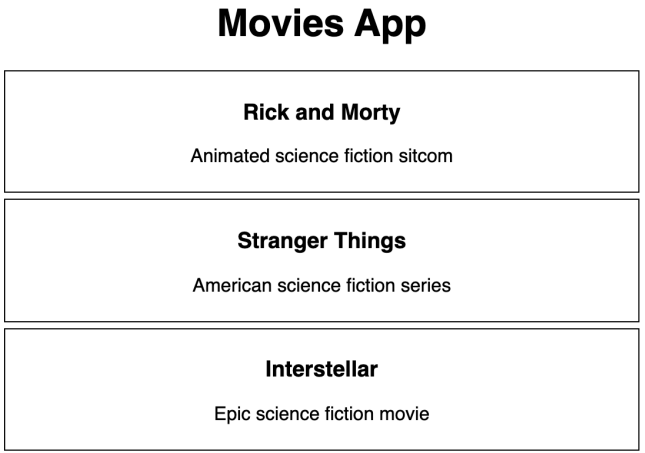
Error Boundary

```jsx
import React from 'react'
import './styles.css'

function ErrorBoundary(props) {
  //fallback UI
  const OopsText = () => (
    <h2>
      Oops, something went wrong... We are doing our best to fix the issue
    </h2>
  )

  // const isEverythingOK = downloadMovies();
  // and do other stuff
  let isEverythingOK = false

  return <>{isEverythingOK ? props.children : <OopsText />}</>
}

export default ErrorBoundary
```

```jsx
import React from 'react'

import Header from './Header'
import ErrorBoundary from './ErrorBoundary'
import MoviesList from './MoviesList'

export default function App() {
  return (
    <>
      <Header />
      <ErrorBoundary>
        <MoviesList />
      </ErrorBoundary>
    </>
  )
}
```

Error Boundary

```
import React from 'react'
import './styles.css'

function ErrorBoundary(props) {
  //fallback UI
  const OopsText = () => (
    <h2>
      Oops, something went wrong... We are doing our best to fix the issue
    </h2>
  )

  // const isEverythingOK = downloadMovies();
  // and do other stuff
  let isEverythingOK = false

  return <>{isEverythingOK ? props.children : <OopsText />}</>
}

export default ErrorBoundary
```



Error Boundary

```
import React from 'react'
import './styles.css'

function ErrorBoundary(props) {
  //fallback UI
  const OopsText = () => (
    <h2>
      Oops, something went wrong... We are doing our best to fix the issue
    </h2>
  )

  // const isEverythingOK = downloadMovies();
  // and do other stuff
  let isEverythingOK = false

  return <>{isEverythingOK ? props.children : <OopsText />}</>
}

export default ErrorBoundary
```

Movies App

Oops, something went wrong... We are doing our best to fix the issue.

HIGHER-ORDER COMPONENTS

Higher-Order Component: Higher-order functions

JavaScript has some of these functions already built in. Some examples of higher-order functions are the following:

.forEach()
This iterates over every element in an array with the same code, but does not change or mutate the array, and it returns undefined.

.map()
This method transforms an array by applying a function to all of its elements, and then building a new array from the returned values.

.reduce()
This method executes a provided function for each value of the array (from left to right).

.filter()
This checks every single element in an array to see whether it meets certain criteria as specified in the filter method, and then it returns a new array with the elements that match the criteria.

Higher-Order Function

```
> getLabel = formatCurrency( '$', '.' );

> getLabel( 1999 )
"$19.99" //formatted value

> getLabel( 2499 )
"$24.99" //formatted value
```

```
const formatCurrency = function (currencySymbol, decimalSeparator) {
  return function (value) {
    const wholePart = Math.trunc(value / 100)
    let fractionalPart = value % 100
    if (fractionalPart < 10) {
      fractionalPart = '0' + fractionalPart
    }
    return `${currencySymbol}${wholePart}${decimalSeparator}${fractionalPart}`
  }
}
```

Higher-Order Component

A **higher-order component (HOC)** is a function that takes a component and returns a new component.

It's a pattern that is derived from React's compositional nature.
We call them **pure components** because they can accept any dynamically provided child component but they won't modify or copy any behavior from their input components.

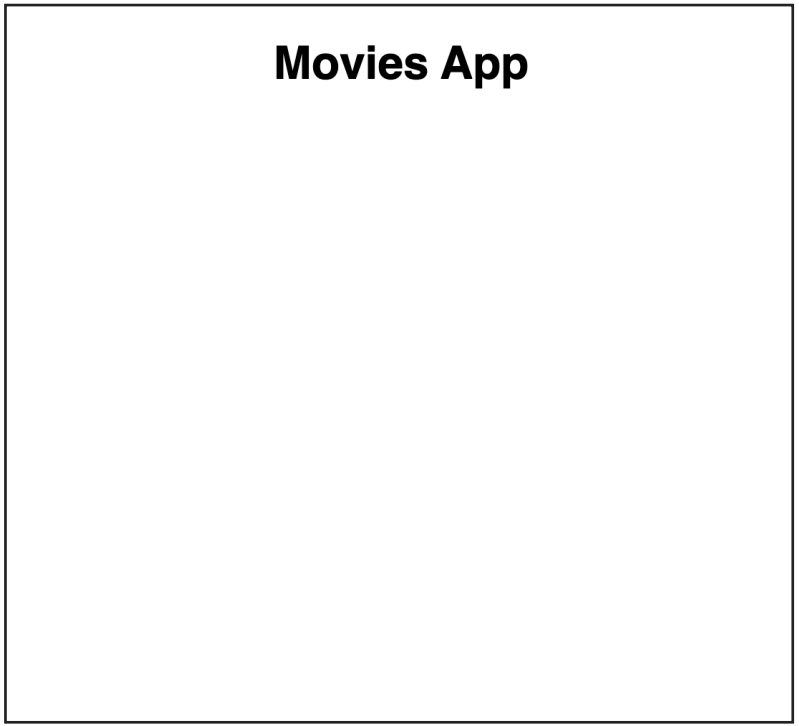
```
const EnhancedComponent = higherOrderComponent(WrappedComponent)
```

HOC can be used for many use cases:

1. Code reuse, logic and bootstrap abstraction.
2. Render hijacking.
3. State abstraction and manipulation.
4. Props manipulation.

Higher-Order Component

Use Case
Verify the status of a user and display proper data



Higher-Order Component

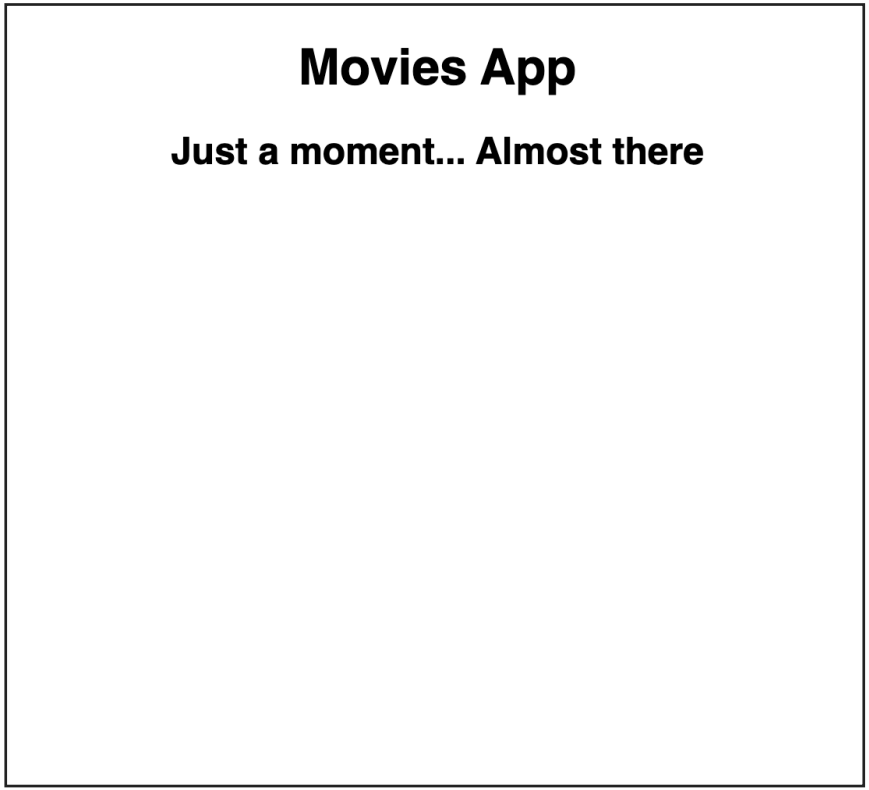
```
import React from 'react'
import './styles.css'

function WithLoading(Component) {
  //fallback UI
  const LoadingIndication = () => <h2> Just a moment... Almost there </h2>

  return function WihLoadingComponent({ isLoading, ...props }) {
    if (!isLoading) return <Component {...props} />
    return <LoadingIndication />
  }
}

export default WithLoading
```

A HOC is structured like a higher-order function:

- It is a component.
- It takes another component as an argument.
- Then, it returns a new component.
- The component it returns can render the original component that was passed to it.

Higher-Order Component

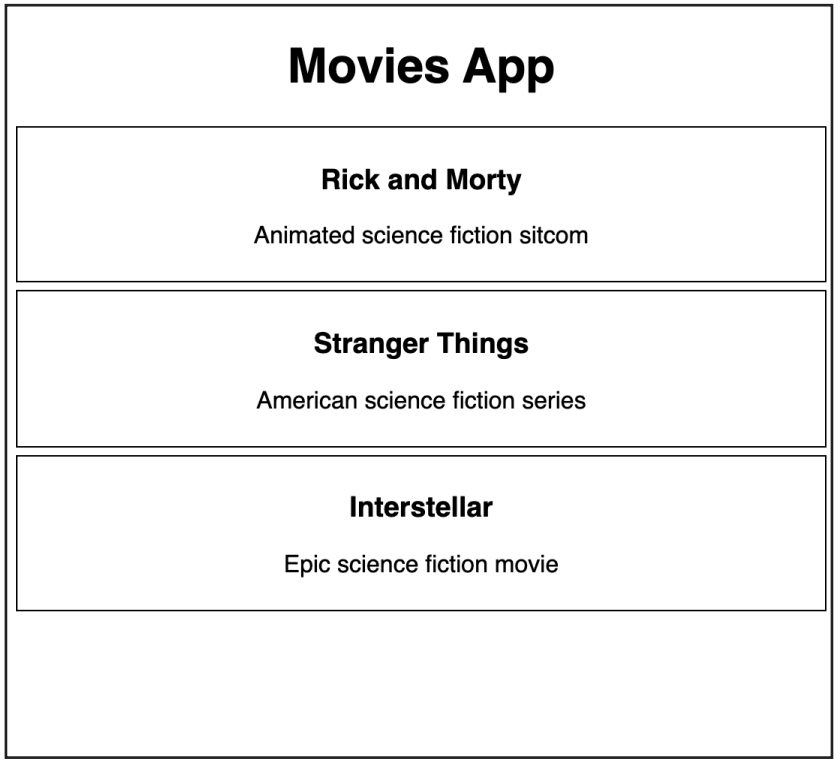
```jsx
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'

import ErrorBoundary from './ErrorBoundary'

import WithLoading from './WithLoading'

const MoviesListWithLoading = WithLoading(MoviesList)

export default function App() {
  return (
    <>
      <Header />
      <ErrorBoundary>
        <MoviesListWithLoading isLoading />
      </ErrorBoundary>
    </>
  )
}
```



Higher-Order Component

*Don’t Mutate the Original Component. Use Composition.

```
import React from 'react'

import Header from './Header'
import MoviesList from './MoviesList'

import ErrorBoundary from './ErrorBoundary'

import WithLoading from './WithLoading'

const MoviesListWithLoading = WithLoading(MoviesList)

export default function App() {
  return (
    <>
      <Header />
      <ErrorBoundary>
        <MoviesListWithLoading isLoading={false} />
      </ErrorBoundary>
    </>
  )
}
```



STRICT MODE

Strict Mode

StrictMode is a tool for highlighting potential problems in an application.
Like **Fragment**, **StrictMode** does not render any visible UI.
It activates additional checks and warnings for its descendants.

Note:
Strict mode checks are run in development mode only; they do not impact the production build.

StrictMode currently helps with:

- Identifying components with unsafe lifecycles
- Warning about legacy string ref API usage
- Warning about deprecated findDOMNode usage
- Detecting unexpected side effects
- Detecting legacy context API
- Additional functionality will be added with future releases of React.

Strict Mode

```
import React from 'react'
import ReactDOM from 'react-dom'

import App from './App'

const rootElement = document.getElementById('root')

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,

  rootElement
)
```

```
▶Warning: Legacy context API has been detected within a strict-mode tree:
    in div (at App.js:32)
    in App (at index.js:7)

Please update the following components: LegacyContextConsumer, LegacyContextProvider

Learn more about this warning here:
https://fb.me/react-strict-mode-warnings
```

APPLICATION STRUCTURE

App Structure

default CRA

Tech Separation

Domain

Deep Domain

historically it so happened that we store components here, business logic here …

COMPONENTS WRAPPING UP

Wrapping up

- Start with Functional Components
- Use <Fragments> instead of s
- Don’t overkill the styles
- Components Shouldn’t Position Themselves
- Styles Should be Abstracted Away
- PropTypes and Strict Mode might help in dev mode
- Props are read only
- Use Layout components
- Apply Parent-Child relationship
- Apply a few Containers
- Higher-Order components will help Loading, etc
- Error Boundaries might rescue from an issue
- Apply a decent app structure
- Composition over Inheritance

THANKS

HOME TASK Q&A