<https://reactjs.org/docs/optimizing-performance.html>

<https://medium.com/myheritage-engineering/how-to-greatly-improve-your-react-app-performance-e70f7cbbb5f6>

<https://www.codementor.io/blog/react-optimization-5wiwjnf9hj>

**Performance Optimization Techniques for React Apps**

### Function/Stateless Components and React.PureComponent

In React, function components and PureComponent provide two different ways of optimizing React apps at the component level.

Function components prevent constructing class instances while reducing the overall bundle size as it minifies better than classes.

On the other hand, in order to optimize UI updates, we can consider converting function components to a PureComponent class (or a class with a custom shouldComponentUpdate method). However, if the component doesn’t use state and other life cycle methods, the initial render time is a bit more complicated when compared to function components with potentially faster updates.

**When should we use React.PureComponent?**

React.PureComponent does a shallow comparison on state change. This means it compares values when looking at primitive data types, and compares references for objects. Due to this, we must make sure two criteria are met when using React.PureComponent:

* Component State/Props is an immutable object;
* State/Props should not have a multi-level nested object.

**Pro Tip:** All child components of React.PureComponent should also be a Pure or functional component.

### Multiple Chunk Files

Your application always begins with a few components. You start adding new features and dependencies, and before you know it, you end up with a huge production file.

You can consider having two separate files by separating your vendor, or third-party library code from your application code by taking advantage of [CommonsChunkPlugin](https://webpack.js.org/plugins/commons-chunk-plugin/" \t "_blank) for webpack. You’ll end up with vendor.bundle.js and app.bundle.js. By splitting your files, your browser caches less frequently and parallel downloads resources to reduce load time wait.

**Note:** If you are using the latest version of webpack, you can also consider [SplitChunksPlugin](https://webpack.js.org/plugins/split-chunks-plugin/" \t "_blank)

### Using Production Mode Flag in Webpack

If you are using webpack 4 as a module bundler for your app, you can consider setting the mode option to **production**. This basically tells webpack to use the built-in optimization:

module.exports = {

mode: 'production'

};

Alternatively, you can pass it as a CLI argument:

webpack --mode=production

Doing this will limit optimizations, such as minification or removing development-only code, to libraries. It will not expose source code, file paths, and [many more](https://medium.com/webpack/webpack-4-mode-and-optimization-5423a6bc597a).

### Dependency optimization

When considering optimizing the application bundle size, it’s worth checking how much code you are actually utilizing from dependencies. For example, you could be using Moment.js which includes localized files for multi-language support. If you don’t need to support multiple languages, then you can consider using [moment-locales-webpack-plugin](https://www.npmjs.com/package/moment-locales-webpack-plugin) to remove unused locales for your final bundle.

Another example is loadash. Let’s say you are only using 20 of the 100+ methods, then having all the extra methods in your final bundle is not optimal. So for this, you can use [lodash-webpack-plugin](https://github.com/lodash/lodash-webpack-plugin" \t "_blank) to remove unused functions.

Here is an extensive list of [dependencies](https://github.com/GoogleChromeLabs/webpack-libs-optimizations) which you can optimize.

### Use React.Fragments to Avoid Additional HTML Element Wrappers

React.fragments lets you group a list of children without adding an extra node.

**class** **Comments** **extends** **React**.**PureComponent**{

render() {

**return** (

<React.Fragment>

<h1>Comment Title</h1>

<p>comments</p>

<p>comment time</p>

</React.Fragment>

);

}

}

But wait! There is the alternate and more concise syntax using React.fragments:

**class** **Comments** **extends** **React**.**PureComponent**{

render() {

**return** (

<>

<h1>Comment Title</h1>

<p>comments</p>

<p>comment time</p>

</>

);

}

}

### Avoid Inline Function Definition in the Render Function.

Since functions are objects in JavaScript ({} !== {}), the inline function will always fail the prop diff when React does a diff check. Also, an arrow function will create a new instance of the function on each render if it's used in a JSX property. This might create a lot of work for the garbage collector.

**default** **class** **CommentList** **extends** **React**.**Component** {

state = {

comments: [],

selectedCommentId: null

}

render(){

**const** { comments } = **this**.state;

**return** (

comments.map((comment)=>{

**return** <Comment onClick={(e)=>{

this.setState({selectedCommentId:comment.commentId})

}} comment={comment} key={comment.id}/>

})

)

}

}

Instead of defining the inline function for props, you can define the arrow function.

**default** **class** **CommentList** **extends** **React**.**Component** {

state = {

comments: [],

selectedCommentId: null

}

onCommentClick = (commentId)=>{

**this**.setState({selectedCommentId:commentId})

}

render(){

**const** { comments } = **this**.state;

**return** (

comments.map((comment)=>{

**return** <Comment onClick={this.onCommentClick}

comment={comment} key={comment.id}/>

})

)

}

}

### Avoiding Props in Initial States

We often need to pass initial data with props to the React component to set the initial state value.

Let's consider this code:

**class** **EditPanelComponent** **extends** **Component** {

**constructor**(props){

**super**(props);

**this**.state ={

isEditMode: false,

applyCoupon: props.applyCoupon

}

}

render(){

**return** <div>

{this.state.applyCoupon &&

<>Enter Coupon: <Input/></>}

</div>

}

}

Everything looks good in the snippet, right?

But what happens when props.applyCoupon changes? Will it be reflected in the state? If the props are changed without the refreshing the component, the new prop value will never be assigned to the state’s applyCoupon. This is because the constructor function is only called when EditPanelComponent is first created.

To quote **React docs**:

*Using props to initialize a state in constructor function often leads to duplication of “source of truth”, i.e. where the real data is. This is because constructor function is only invoked when the component is first created.*

**Workaround:**

1. Don't initialize state with props which can be changed later. Instead, use props directly in the component.

**class** **EditPanelComponent** **extends** **Component** {

**constructor**(props){

**super**(props);

**this**.state ={

isEditMode: false

}

}

render(){

**return** <div>{this.props.applyCoupon &&

<>Enter Coupon:<Input/></>}</div>

}

}

1. You can use componentWillReceiveProps to update the state when props change.

**class** **EditPanelComponent** **extends** **Component** {

**constructor**(props){

**super**(props);

**this**.state ={

isEditMode: false,

applyCoupon: props.applyCoupon

}

}

*// reset state if the seeded prop is updated*

componentWillReceiveProps(nextProps){

**if** (nextProps.applyCoupon !== **this**.props.applyCoupon) {

**this**.setState({ applyCoupon: nextProps.applyCoupon })

}

}

render(){

**return** <div>{this.props.applyCoupon &&

<>Enter Coupon: <Input/></>}</div>

}

}

**References and Related Articles:**  
[ReactJS: Why is passing the component initial state a prop an anti-pattern?](https://stackoverflow.com/questions/28785106/reactjs-why-is-passing-the-component-initial-state-a-prop-an-anti-pattern),  
[React Anti-Patterns: Props in Initial State](https://medium.com/@justintulk/react-anti-patterns-props-in-initial-state-28687846cc2e)

### Memoize React Components

[Memoization](https://en.wikipedia.org/wiki/Memoization) is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again. A memoized function is usually faster because if the function is called with the same values as the previous one then instead of executing function logic it would fetch the result from cache.

Let's consider below simple stateless UserDetails React component.

**const** UserDetails = ({user, onEdit}) => {

**const** {title, full\_name, profile\_img} = user;

**return** (

<div className="user-detail-wrapper">

<img src={profile\_img} />

<h4>{full\_name}</h4>

<p>{title}</p>

</div>

)

}

Here, all the children in UserDetails are based on props. This stateless component will re-render whenever props changes. If the UserDetails component attribute is less likely to change, then it's a good candidate for using the memoize version of the component:

**import** moize **from** 'moize';

**const** UserDetails = ({user, onEdit}) =>{

**const** {title, full\_name, profile\_img} = user;

**return** (

<div className="user-detail-wrapper">

<img src={profile\_img} />

<h4>{full\_name}</h4>

<p>{title}</p>

</div>

)

}

export default moize(UserDetails,{

isReact: true

});

This method will do a shallow equal comparison of both props and context of the component based on strict equality.

If you are using **React V16.6.0** or greater version, then you can use React.memo and rewrite the above code like:

**const** UserDetails = ({user, onEdit}) =>{

**const** {title, full\_name, profile\_img} = user;

**return** (

<div className="user-detail-wrapper">

<img src={profile\_img} />

<h4>{full\_name}</h4>

<p>{title}</p>

</div>

)

}

export default React.memo(UserDetails)

The following links can be useful if you would like to know more about some of the topics from this article in more detail:

* [How I wrote the world’s fastest React memoization library](https://itnext.io/how-i-wrote-the-worlds-fastest-react-memoization-library-535f89fc4a17)
* [Memoize React components](https://medium.com/@planttheidea/memoize-react-components-33377d7ebb6c)  
  [theKashey/react-memoize](https://github.com/theKashey/react-memoize)
* [How to use Memoize to cache JavaScript function results and speed up your code](https://medium.freecodecamp.org/understanding-memoize-in-javascript-51d07d19430e)
* [NPM: Moize](https://www.npmjs.com/package/moize)
* [Higher Order Functions](http://eloquentjavascript.net/05_higher_order.html#h_xxCc98lOBK)

