

# **Understanding State and Lifecycle in React.js**

React.js is a popular open-source JavaScript library for building user interfaces. One of the key features of React.js is its ability to manage state and lifecycle of components. This document will provide you with a deeper insight into what these concepts are, how to manage them, and why they are important in building effective React.js applications.



**by Curious Coder ([CodeWithCurious.com](http://CodeWithCurious.com))**

# The Importance of State

In React.js, state refers to the collection of data that determines the behavior of a component. It is critical to understand state as it plays a crucial role in updating the interface based on user input. Depending on the state of a component, its output can change dynamically. This section will provide an overview of the features and use of state.

## Features of State

- Changes dynamically
- Holds data
- Enables re-rendering of components

## Use of State

Using state in React allows for the creation of dynamic and responsive user interfaces. It is the simplest way to handle and store data in a React application.

# The Lifecycle of React Components

The lifecycle of a component in React refers to the series of phases it goes through from initialization to destruction. These phases, called 'lifecycle methods,' are invoked in a particular order. Understanding the lifecycle of React Components is crucial in building effective React.js applications.

Phase	Description
Mounting	Component is initialized and inserted into the DOM
Updating	Occurs when a component's state or props change
Unmounting	When a component is removed from the DOM

# Managing the Component State

In this section, we will look at how to set, read, and update the state of a component. We will also discuss the use of React hooks and how they simplify state management.

## 1 Set State

A setState method is called to set a new state for the component. This will update the interface to reflect the new state.

## 2 Read State

To access the current state of a component, the this.state object can be used.

## 3 Update State

The new state can be set by calling the setState method, which updates the state by merging the new state with the existing one.

# Lifecycle Methods

The lifecycle methods in React.js are methods that are executed at different points in the lifecycle of a component. These methods are useful for enhancing and modifying the behavior of components. In this section, we will look at some popular lifecycle methods.

## **componentDidMount()**

This method is invoked after the component has been mounted to the DOM. It is used to load initial data that the app needs for smooth operation, such as data from an API.

## **componentDidUpdate()**

This method is called every time the component is updated. This method is useful for setting changes to the global state.

## **shouldComponentUpdate()**

This method is called before rendering the component. It is used to determine if the component needs to be re-rendered, thus optimizing the performance of the app.

# Updating State vs. Props

Knowing when to use state vs. props is crucial to building effective React applications. Both props and state are used in React.js to control what appears on the screen. However, they are used for different purposes.

Use	Props	State
Can change during Component Lifecycle?	No	Yes
Declared in the parent or child component?	Parent	Within Component
Changes	Immutable	Mutable

# Updating State

There are several ways to update state in React.js: through `setState`, setting state to self, and using callback functions. In this section, we will explore each of these methods briefly.

## **useState**

Allows you to merge a new state with the existing state of a component. This is preferred as it avoids direct state manipulation.

## **Setting state to self**

Used when you want to use the current state to derive the new state of the component. However, this method can lead to errors, and it is recommended to use functional `useState` instead.

## **Using callback functions**

Encapsulates the setting of state in a callback function, allowing you to access the current and new states of the component.

# Mounting and Unmounting Components

React.js provides methods that are called during the mounting and unmounting of components. These methods can be overridden and used to define custom behaviors. In this section, we will look at the different methods available to add functionality to components during these phases.

Method	Description
componentDidMount	Called after the component has finished rendering. Used to load data from an API or to set up event listeners.
componentWillUnmount	Called before the component is removed from the DOM. Used to clean up component resources.

# Conclusion

State and Lifecycle are two of the fundamental concepts in building React.js applications. By understanding the use of these techniques, you can create custom and responsive user interfaces, and build more effective and maintainable applications.

# JSX and Components in React

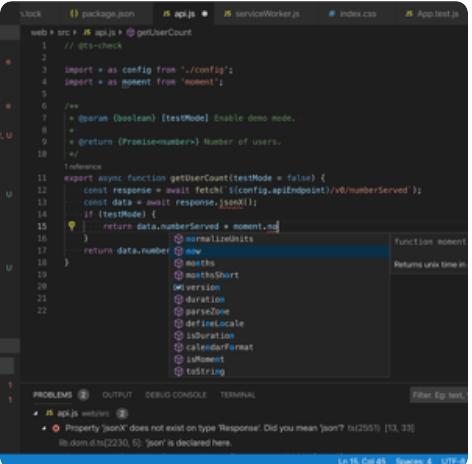
In this presentation, we'll explore the world of JSX and Components in React and learn how they can help us build powerful web applications.

by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))



```
index.ts x .env.local x _app.tsx x
ts/blog main / 9 Δ 1 import { useEffect } from 'react'; 8.23 kB (gzip: 3.33 kB)
2 import Head from 'next/head';
3 import type { AppProps } from 'next/app'; 5.11 kB (gzip: 1.54 kB)
4 import { ApolloProvider } from '@apollo/client'; 123.67 kB (gzip: 3.67 kB)
5 import { ThemeProvider } from '@material-ui/core/styles';
6 import CssBaseline from '@material-ui/core/CssBaseline';
7 import { Container } from '@material-ui/core'; 63.32 kB (gzip: 1.64 kB)
8 import { useApollo } from '../graphql/client';
9
10 import { lightTheme, darkTheme } from '../utils/theme';
11 import useLocalStorage from '../hooks/useLocalStorage';
12
13 import NavBar from '../components/NavBar';
14
15 function App({ Component, pageProps }: AppProps) {
16   const [currentTheme, setCurrentTheme] = useLocalStorage('theme');
17   const apolloClient = useApollo(pageProps.initialState);
18
19   useEffect(() => {
20     const jssStyles = document.querySelector('#jss-1');
21     if (jssStyles) {
22       jssStyles.parentElement.removeChild(jssStyles);
23     }
24   }, [deps]);
25
26   return (
27     <>
28       <Head>
29         <title>ECU-DEV</title>
30         <meta name="viewport" content="width=device-width, initial-scale=1" />
31       </Head>
32       <ThemeProvider theme={currentTheme}>
33         <ApolloProvider client={apolloClient}>
34           <CssBaseline />
35           <Container>
36             <Component {...pageProps} />
37           </Container>
38         </ApolloProvider>
39       </ThemeProvider>
40     </>
41   );
42 }
```

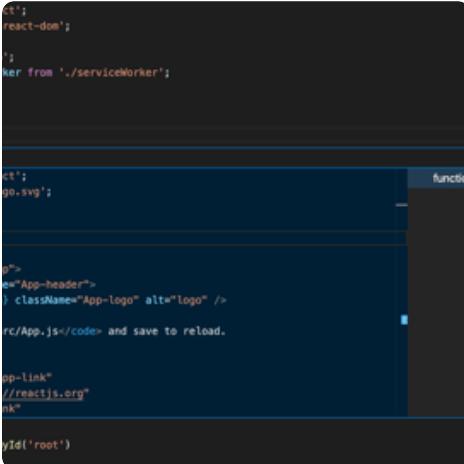
# What is JSX?



A screenshot of a code editor showing a file named `api.js`. The code uses the `moment` library to handle dates. A tooltip for the `moment` module is open, showing its various methods like `normalizeUnits`, `now`, `format`, `formatShort`, `version`, `duration`, `parseZone`, `defineLocale`, `isDuration`, `createFormat`, `startOf`, and `toDateString`.

## Javascript Syntax

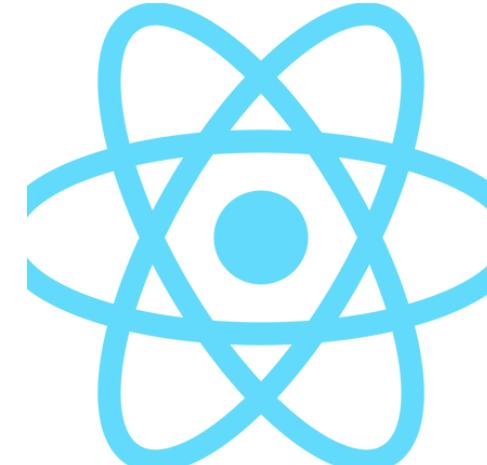
JSX is a syntax extension for JavaScript that allows us to write HTML-like code that can be transformed into JavaScript.



A screenshot of a code editor showing a file named `App.js`. It contains JSX code where HTML-like tags like `<div>` and `<p>` are used to structure the UI. The code also includes imports for `react-dom` and `react` from the `./serviceWorker` file.

## JSX Syntax Example

With JSX, we can write code that looks like HTML, and render it using React.



## JSX + React

JSX is a key feature of React, and is used to build reusable components and UI elements.

# Benefits of Using JSX

## 1 Improved Code Readability

JSX helps us write code that's easier to read and understand, especially when working with nested components.

## 2 Faster Development

By allowing us to write HTML-like code, JSX makes it faster to build and prototype UI components.

## 3 Better Performance

Since JSX helps us avoid unnecessary DOM updates, it can lead to improved performance in our applications.



# Introduction to Components

## What are Components?

Components are reusable UI elements that can be composed together to build complex user interfaces.

## Why Use Components?

By breaking our UI down into smaller components, we can make it easier to reason about and maintain our code.

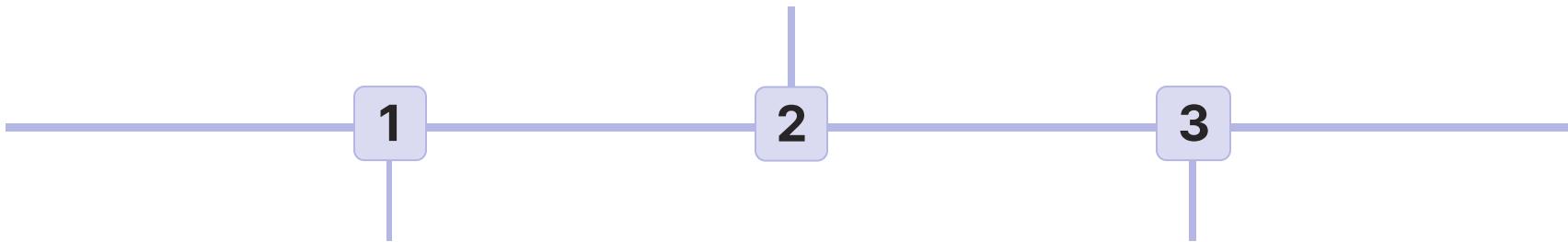
## Types of Components

There are two types of components in React: Class Components and Functional Components.

# Class Components vs. Functional Components

## Functional Components

Functional components are defined using JavaScript functions, and are simpler and easier to test than class components.



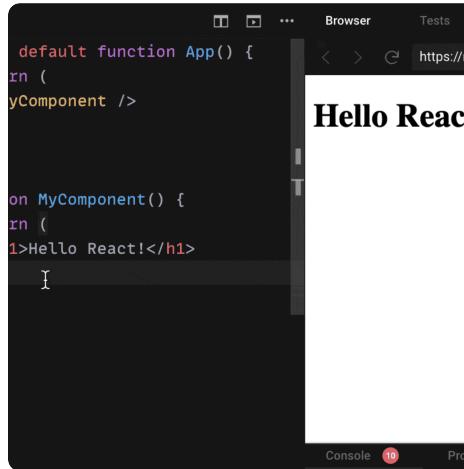
### Class Components

Class components are defined using ES6 Classes, and can hold state and lifecycle methods.

### Choosing the Right Component Type

When deciding between class and function components, consider the complexity of your UI and whether you need to manage state.

# Props and State in Components



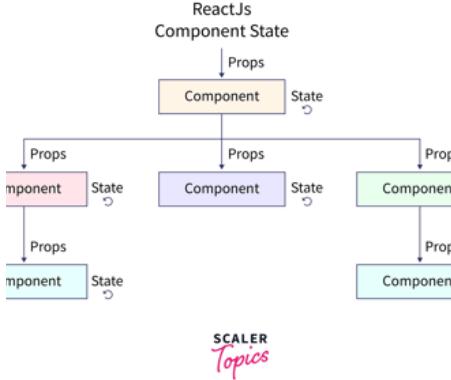
A screenshot of a code editor showing a React component structure. The code defines an `App` component that contains a `MyComponent` component. The `MyComponent` component displays the text "Hello React!".

```
default function App() {
  return (
    <MyComponent />
  );
}

function MyComponent() {
  return (
    <h1>Hello React!</h1>
  );
}
```

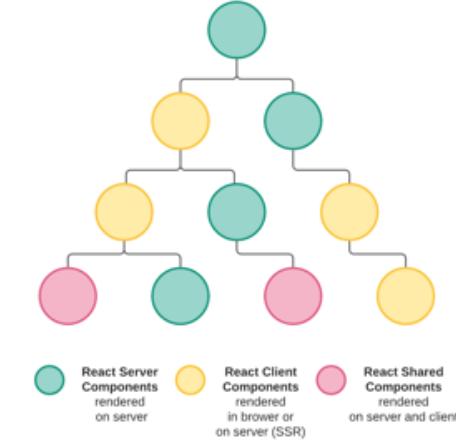
## Props

Props are read-only values that are passed from a parent component to a child component, and can be used to customize component behaviour.



## State

State is an internal value that can be updated by a component, and is used to manage user interface interactions and component behaviour.



## Props vs. State

Props and state are both important concepts in React, and understanding how they work is essential for building robust and maintainable components.

# Rendering Components

## Creating Components

To create a new component, define a function or class that returns some JSX.

## Rendering Components

To use a component, render it in another component or in the main application.

## Composing Components

By combining our components together, we can build complex user interfaces made up of simple, reusable building blocks.

# Conclusion and Resources for Learning More

## 1 What We've Learned

Today, we've learned what JSX is, the benefits of using it, and how to create and use components in React.

## 2 Next Steps

If you want to learn more about React components and JSX, check out the official React documentation or some online tutorials.

## 3 Resources

React Documentation, FreeCodeCamp, Udemy, PluralSight and Codecademy are some platforms for React JS learning.



# Introduction to React.js

React.js is a powerful JavaScript library used to create interactive and dynamic user interfaces. It enables developers to build complex applications with ease and is a popular choice for both beginners and seasoned web developers.



by Curious Coder ([CodeWithCurious.com](http://CodeWithCurious.com))

# Key Features of React.js

## Virtual DOM

React uses a Virtual DOM, a lightweight replica of the browser's DOM, which enables fast and efficient rendering.

## Reusable Components

Components in React enable developers to create reusable interfaces, which can be easily modified and reused throughout the application.

## Unidirectional Data flow

React features a unidirectional data flow, which makes it easier to debug and understand the application's data flow.

## JSX Syntax

React uses JSX - JavaScript syntax extension, which enables the creation of component hierarchies in a readable and streamlined way.

# How React.js Compares to Other Frameworks



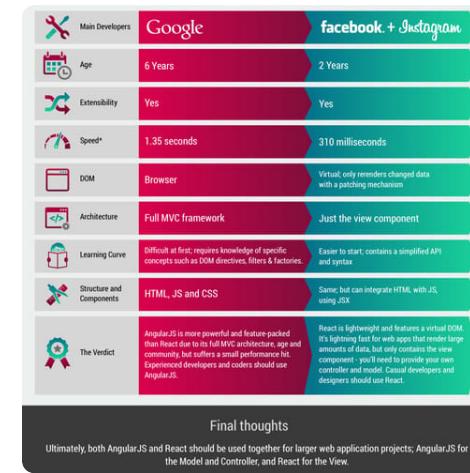
## React.js vs Angular

While both frameworks are popular, Angular is based on a MVC pattern while React.js uses an MVVM pattern. React tends to be more flexible and easier to learn thanks to its component-based approach.

	ANGULAR	REACT
Lead	Miško Hevery	Jordan Walke
Release Date	May 2016	March 2013
Script	TypeScript	JavaScript
Complexity	high	average
Implementation	very complex	complex

## React.js vs Vue.js

Both frameworks are popular but Vue.js is considered easier to learn and use, making it a good choice for small to medium-sized applications. However, React is more popular and is often preferred for large-scale applications.



## React.js vs jQuery

React.js is a full-fledged front-end library, whereas jQuery is a JavaScript library that focuses on DOM manipulation and event handling. React.js is ideal for building complex single-page applications, while jQuery is better suited for smaller websites.

# Using Components in React.js

## 1 Class Components

These components are JavaScript Classes that extends React Component and have their own state and lifecycle methods.

## 2 Functional Components

These components are simpler and more lightweight since they don't have state and lifecycle methods. They only take in props as parameters and render the component.



# React.js Lifecycle Methods

The render method is called each time a component is updated, it's the main method that uses JSX to create your components.

**render()**

1

2

3

**constructor()**

This is the first method that is called when a component is created, and it's used for initializing state and binding event listeners.

**componentDidMount()**

This method is called after the component is mounted and can be used to execute AJAX calls, modifications to state or any other setup that requires changes to the DOM.

# How to Get Started with React.js

## Step 1

Install Node.js, which comes with npm, the package manager for JavaScript libraries such as React.js.

## Step 2

Create a new project using `create-react-app`, the official command-line tool for creating React.js applications.

## Step 3

Start building your application, by creating components, handling state, and adding other libraries or resources if needed.

# Conclusion

React.js is a powerful and popular front-end library that enables developers to build complex web applications with ease. Its component-based architecture, unidirectional data flow, and JSX syntax make it a top choice for both beginners and seasoned developers.

# Handling Events in React.js

In this article, we'll explore how to handle events with React.js, one of the most popular JavaScript libraries for building web applications. We'll cover everything from binding events to passing arguments and preventing default behaviours. Let's dive in!



by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))

# Introduction to Handling Events

## What are events?

In web development, an event refers to anything that happens on a web page, such as when a button is clicked or a form is submitted. In React, you can handle these events with event handlers.

## Why is event handling important?

Event handling is crucial in creating dynamic web applications. By responding to events with custom logic, you can create user interactions that make your application feel more responsive and intuitive.

# Binding Events in React

When you want to bind an event to a component in React, you need to define a function that will be called when the event is triggered. The function is then bound to the component using the "this" keyword.

- React Documentation

# Event Handlers in React

Event	Handler
onClick	handleClick
onChange	handleChange
onSubmit	handleSubmit

Here are some examples of common events and their respective handlers in React. Keep in mind that event handlers should be defined as class methods, not within the render method.

# **Understanding Synthetic Events in React**

Synthetic events are a cross-browser wrapper around the native browser events. They have the same interface as native events, including `stopPropagation()` and `preventDefault()`, but work identically across all browsers. In addition, synthetic events can be reused to enhance performance.

# Passing Arguments to Event Handlers in React

Passing arguments to an event handler is essential when you want to perform actions on the data associated with the event. The most common way to do this is by using arrow functions to create a closure around the event handler function.

## 1 Example:

```
handleClick = (id) => () => console.log(id);  
<button onClick={this.handleClick(itemId)}>Click me</button>
```

# Preventing Default Behaviour in React

By default, every HTML element has a predefined behaviour. For example, if you click a link, the browser will navigate to the URL specified in the href attribute. In some cases, you may want to prevent this default behaviour to perform custom actions instead.

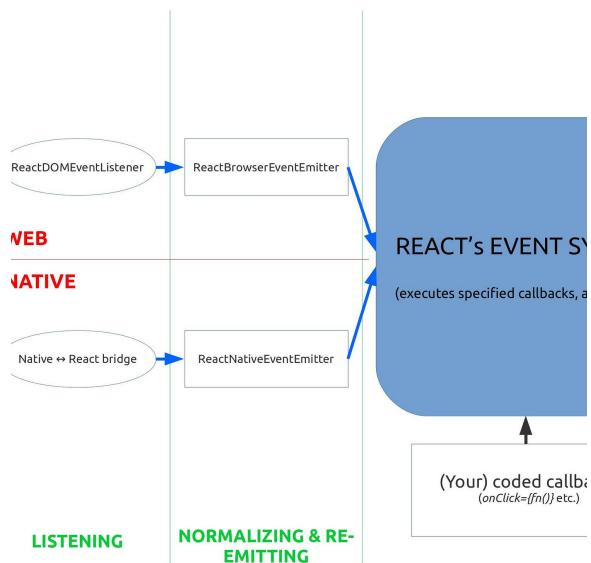
## Method 1: Using `event.preventDefault()`

```
handleClick = (event) => {
  event.preventDefault(); console.log('Link
  clicked!');
```

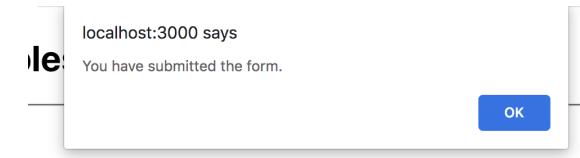
## Method 2: Using a button element

```
<button type="button" onClick=
{this.handleClick}>Click me</button>
```

# Example of Handling Events in React



Example of a button click event



Example of a form submission event

Here are a couple of examples of how you can handle events in React. These are simple examples, but they demonstrate the core concepts of event handling in React.

# Lists and Keys in React.js

Learn how to create lists in React.js and add unique keys to list items. This guide covers the benefits of using keys, best practices for using keys and how to update and reorder lists in React.js.



by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))

# **Understanding Lists in React.js**

Lists in React.js are a great way of rendering data. In this section, we go over the basics such as what lists are and how to create them. We'll discuss the different types of lists and how to render them to the DOM. Images can display various types of lists as well, such as ordered and unordered.

# The Importance of Unique Keys in Lists

A key is a special string attribute that's included when creating lists of elements. It's important for performance reasons and React will warn you if your keys are not unique. A key allows the framework to efficiently re-render only the items that have changed, instead of the entire list.

# Creating Lists with Keys in React.js

In this section, we dive deep into creating lists with keys. We'll start with the basics of adding keys to avoid any future issues. Additionally, we'll go over some of the more advanced concepts such as how to create dynamic keys and how to create keys based on an item's ID. By the end of this section, you'll have a good understanding of the importance of keys in creating lists.

# The Benefits of Using Keys in React.js

## Efficiency

Keys help React to update and re-render only the elements that have changed, thus improving application performance and reducing development time.

## Clarity and Readability

Lists with unique keys are easier to read and more organized, making it easier to identify issues and debug code.

## Better User Experience

With keys, data is organized such that your users aren't confused by changes to your lists, which leads to a better user experience.

# Best Practices for Using Keys in React.js

## 1 Use a Unique ID or Index

Create unique keys for each list item using an ID or index, which is less error-prone and makes debugging more comfortable.

## 2 Avoid Using Random Values

Using random values may create problems for performance and makes it challenging to track down issues when they occur.

## 3 Avoid Using the Array Index as Key

Using an array index as a key should be avoided whenever possible. Using an index as a key can lead to bugs, especially when you delete or move list items.

# **Updating and Reordering Lists in React.js**

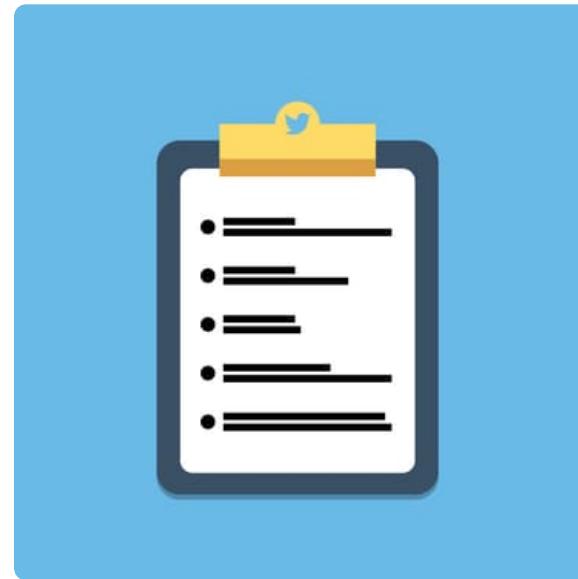
React.js makes it easy to update an existing list or reorder it. We'll go over several strategies for optimizing and maintaining the list data. This section will also discuss how to handle data changes and filter your lists efficiently. By the end of this section, you'll be able to create lists that can easily handle data changes.

# Conclusion and Key Takeaways



## Unique keys are important

Using unique keys in your lists is key to optimizing performance and maintaining a clean, organized codebase. Make sure you follow the best practices we've discussed and use keys for all your lists.



## Lists are powerful

By using lists, you can effectively render data in your React.js applications. Make sure to understand the nuances of rendering lists in React, so you can take full advantage of this powerful feature.

# **Styling and CSS in React.js**

In this guide, you will learn about the different ways to style components in React.js. We will explore both traditional and modern approaches to styling and offer best practices for how to implement them.



**by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))**

# **Introduction to React.js and CSS**

## **What is React.js?**

React.js is an open-source JavaScript library for building user interfaces. It was developed by Facebook and has since become one of the most popular front-end libraries in use today.

## **Why is CSS important in web development?**

CSS is a crucial component of web development because it allows developers to style and modify the appearance of HTML and JavaScript elements. With CSS, developers can create visually pleasing and engaging web pages.

# **Different ways to style components in React.js**

There are several ways to style components in React.js, each with its advantages and disadvantages.

The traditional method used CSS files, but newer approaches such as inline styling with JSX and CSS modules have gained popularity in recent years. CSS-in-JS libraries like styled-components provide a third approach.

# Inline styling with JSX

Inline styling with JSX is a popular choice because it allows developers to apply styles directly to individual components within their code. This makes it easy to customize styles for individual components without affecting other parts of the project.

# CSS modules in React.js

CSS modules are a recent addition to React.js that offer a way to modularize CSS styles. By using unique class names for each component, developers can avoid naming conflicts and create a more organized codebase.

# **CSS-in-JS libraries like styled-components**

Styled-components is one of the most popular CSS-in-JS libraries available for React.js. It allows developers to write CSS directly in their JavaScript code and has a variety of useful features, including scoped styles, theming, and server-side rendering.

# Best practices for styling in React.js

## 1 Keep styles separate from logic

Avoid mixing your styles with your logic to make them more reusable and easier to maintain.

## 2 Be mindful of performance

Large CSS files can slow down your application. Use techniques like code splitting and lazy loading to ensure that only necessary styles are loaded.

## 3 Use CSS-in-JS for larger projects

If your project grows in size, consider using a CSS-in-JS library like styled-components to keep your styles organized and scalable.

# A final word

Styling is an important part of modern web development, and React.js has many options available to simplify the process. By following best practices and choosing the right approach for your project, you can create beautiful and performant user interfaces that will engage your users and enhance their experience.

# React Router: The Ultimate Guide

If you're building a modern single-page web application with React, you need React Router. In this guide, we'll go through everything you need to know to get started, including defining routes, using links, handling URLs, and debugging common errors.



by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))

# Getting started with React Router

React Router is a powerful library that allows you to add URL routing to your React applications. Whether you're building a small app or a large complex project, router will help you manage your routes and create a seamless user experience. To get started, install React Router using npm or yarn.

# Defining routes with <Route>

## Route Component

The route component is the key to defining your routes in React Router. You use it to specify which component to render when the URL matches a certain path. You can also define a default route.

## Path matching

React Router uses a powerful system for matching URLs with specific routes. You can include parameters in your URL and use regular expressions for more complex matching. You can also use query strings and add custom logic to your route components.

# Using `<Link>` to navigate

Adding links is easy with the `<Link>` component. You specify the URL as the `to` prop and React Router will handle the navigation for you. You can also style your links differently based on the current URL.

# Using URL parameters

Path	Component	Params
/users/:id	UserDetails	id
/products/:category/:name	ProductDetails	category, name

URL parameters allow you to include dynamic data in your URLs, such as user IDs or product names. You can access those parameters in your route components using the `match` prop. You can also specify optional parameters and default values.

# **Nested routes and route matching**

Nested routes allow you to define child routes that inherit the URL parameters of their parent routes. This can be useful for creating hierarchies and nested content. You can also use multiple route components to match different URLs, and handle fallback routes.

# Programmatically navigating to a route

Sometimes you want to navigate to a route programmatically, for example after a form submission. You can use the `history` object provided by React Router to do this. You can also pass state data along with your route transitions.

# Common errors and how to debug them

Error: You should not use <Route> outside a <Router>

One of the most common errors when using React Router is forgetting to wrap your application with a <Router> component. This is the container that wraps all your routes and provides the history and location props that route components use. Make sure you add the Router component at the top level of your app.

# Forms and Validation in ReactJs

In this article, we'll explore the fundamentals of Forms and Validation in ReactJs. We'll dive into the different ways you can handle form submission and validation using popular libraries and hooks.



by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))

# Introduction to Forms in ReactJs

Handling forms is a common requirement for any web application. In ReactJs, forms are controlled components meaning that the component state manages the input values rather than the DOM. This section will provide an overview of form handling in ReactJs.

## Uncontrolled Components

Uncontrolled Components are traditional HTML form controls. ReactJs retrieves their current values using a ref. These components feel more natural and browser-native to some developers.

## Controlled Components

Controlled components are React components that provide all of their form data through props to the parent component. This is React's recommended approach to handling form data.

# Form elements in ReactJs

Forms in ReactJs are composed of various form elements that differ in their behavior and can be customized. In this section, we'll explore the basic and complex form elements included in ReactJs and their distinctive features.

## Input

The Input component is a basic form element that receives most user inputs. It can accept a variety of attributes such as 'type', 'value', and 'placeholder' and emit various events such as 'onChange', 'onBlur', and 'onKeyDown'.

## Select

The Select component provides a dropdown list of selectable items. React wraps the native Select component in two other components, the Option component to wrap each option and the OptGroup component to group the options.

## Checkbox

The Checkbox component is a basic form element that is used for selecting one or more options from a set of options. ReactJs provides a rich interface to customize Checkbox functionality.

# Form Submitting in ReactJs

Submitting forms in ReactJs can be achieved through a variety of methods, such as sending an HTTP POST request, interfacing with an external API, etc. In this section, we'll explore how to handle form submission in ReactJs.

Using Formik, developers can have an opportunity to handle and analyze complex forms efficiently. It abstracts away many of the inconveniences associated with handling form submission and provides a means to incorporate complex form validation without having to reinvent the wheel.

# Validation using Formik library

Invalid form data can cause issues for your application and your users. The Formik library provides a simple means of validating users' inputs and displaying error messages.

## 1 Formik Yup validation Schema

Formik uses the Yup library to handle validation. Formik Yup validation schema is a powerful and flexible way to validate user inputs. It offers a simple schema definition via a chainable interface and a comprehensive list of validation rules.

## 2 Formik Helper Functions

Formik provides helper functions to help you test and validate form inputs. It simplifies handling onSubmit and onSubmitting events and provides many additional features to assist with rendering form state, form events, and errors.

# Handling errors and displaying error messages

Error messages help users know why their input has been rejected. The Formik library provides a simple interface to create and display form errors. In this section, we'll explore how to manage errors and display them properly.

## Managing error on a field

Formik provides tools to set, remove or access data from individual form fields. To display an error message, we can associate each error message with its corresponding field using the 'errors' property of Formik.

## Displaying an Error Message

In Formik, a touched property is set to true if the user has interacted with the field. By default, error messages are only displayed if the corresponding field is touched, and an error exists for the field.

# Form Validation using React Hooks

The React Hooks library provides a simple interface to share stateful logic between different components of your application. It is an efficient way to handle data and lifecycle operations. In this section, we'll explore form validation using React Hooks and how to apply them to different areas of your application.

## 1 useState and useEffect Hooks

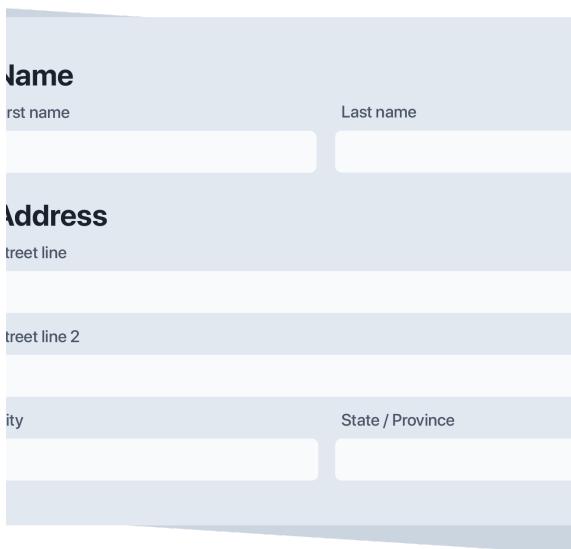
The useState and useEffect Hooks provide you with a convenient way to manage component state and perform data side effects. They're perfect for managing form validation and can be used in combination with other Hooks or traditional React components.

## 2 useForm Hook

The useForm Hook is a custom Hook that simplifies working with a form. It abstracts away many of the inconveniences associated with individual form fields and provides various functions to interact with form data.

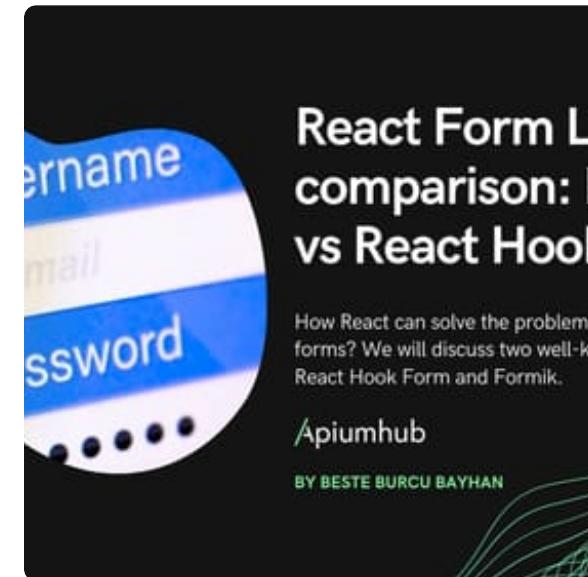
# Conclusion and Key Takeaways

Handling forms and validation in ReactJs can seem daunting, but it doesn't have to be. With ReactJs libraries like Formik and Hooks, you can easily create, validate, and manage your data. In this article, we've covered the basics of form handling and validation and explored various methods to make the process simpler. Remember to keep your design concise, engage users, and make sure they get answers to their questions efficiently.



## Forms in ReactJs

Understand the basic and complex form elements in ReactJs. Learn how to manage data and control user inputs to improve application performance and simplify data handling.



## Formik Library

Learn how to use Formik to manage form data and validation easily. Formik offers a simple schema definition via a chainable interface and a comprehensive list of validation rules.

# State Management in React.js

In this document, we will explore the ins and outs of state management in React.js. We will cover everything from what React.js is to how to use the useState hook. Dive into this exciting topic and learn how to make your React applications more powerful than ever!



by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))

# **What is React.js?**

React is a popular JavaScript library for building user interfaces that was developed by Facebook. React allows for building reusable UI components and using declarative views in your code. It makes it easy to create interactive and dynamic web applications with ease.

# Introduction to state management

## What is state?

State represents the data that your app is currently working with. It can be thought of as a snapshot of your application at any given time.

## Why is state management important?

State management is critical because it allows you to keep track of the current state of your application and update it over time based on user input or other factors. It's essential to keep your application running smoothly and efficiently, especially when dealing with large amounts of data.

## What are the benefits of good state management?

Good state management allows your application to be more maintainable, scalable, and testable. By efficiently managing state, you can ensure that your application is modular, reusable, and easy to maintain.

# **State Management in Class Components**

Class components have been the traditional way of implementing state management in React before the introduction of the hooks API. The state in class components can be modified, mounted and unmounted with component life cycles methods such as `componentDidMount()` and `componentWillUnmount()`

# State Management in Functional Components

```
App extends Component {  
  constructor(props) {  
    super(props);  
    this.state = [ state1: "", state2: "" ]  
  
  componentDidMount() {  
    //TODO: API calls here...  
  
  componentWillMount() {  
  
  }  
  
  render() {  
    return (  
      <div className="App">  
        <header className="App-header">  
          <img src={logo} className="App-logo" alt="logo" />  
          <h1 className="App-title">Welcome to React</h1>  
        </header>  
        <p className="App-intro">  
          To get started, edit <code>src/App.js</code> and save to  
          reload.  
        </p>  
      </div>  
    );  
  }  
  
  default App; |
```

Functional components in React can also manage their state using hooks, including the useState hook. Hooks make it easier to manage state in functional components, making the code more organized and easier to read.

# Using the useState Hook

- The useState is a hook that is used to manage state in React Functional Components
- The hook can handle both single and multiple states.
- Here is an example of how to use useState: `const [count, setCount] = useState(0);`

1

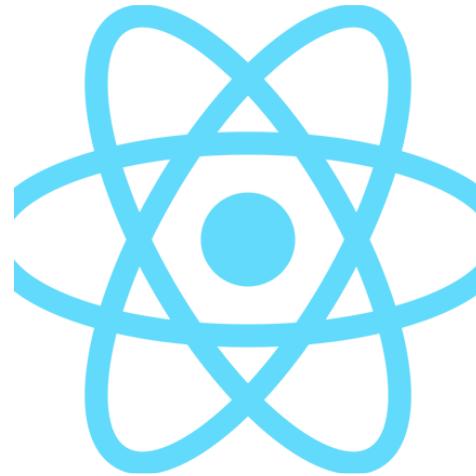
## Key points to keep in mind:

The useState hook returns an array containing the current state value and a function for updating the state. You can use this state variable and function to update the UI based on user input or other factors.

# Best practices for state management in React.js

Best Practice	Description
Keep state as simple as possible	Avoid putting too much data in state as it may cause performance issues. Store only the data that is essential for rendering the UI.
Use the right state management tool for each situation	Context API, Redux are alternatives to manage and share the state globally across all components.
Carefully name your state variables and their update functions	This can help make your code more readable, maintainable and easy to understand.

# Conclusion



React.js is a powerful library for building user interfaces, and managing state is a crucial aspect of creating your applications. By following best practices and choosing the right tools, you can make your React applications more efficient, scalable, and easier to maintain. Start implementing these state management concepts in your next project and watch your code soar to new heights!

# Deployment and Production Considerations in ReactJS

ReactJS deployment involves considerations like performance monitoring, continuous integration and testing, security, and more. In this document, we'll go over these considerations in detail, with tips and best practices to help you deploy your ReactJS app with ease.



by Curious Coder ([CodeWithCurious.com](http://CodeWithCurious.com))

# Deployment Options

Deploying your ReactJS app to production requires choosing the right deployment option. There are several services and tools available to deploy your app, from manual deployment to automated deployment. Deploying manually requires more work, but allows for more control, while automated deployment means faster and error-free deployments.

## Manual Deployment

- Deploying using FTP or SSH
- Requires manual configuration
- More control with the deployment process

## Automated Deployment

- CI/CD tools
- Fast and error free deployment
- Uses pre-configured deployment scripts

# Environment Variables

Using environment variables in your ReactJS app allows you to store runtime configuration settings, such as API endpoints, secrets, and other sensitive data. This keeps your code clean, secure, and reusable, and allows you to switch between development, staging, and production environments without changing the code.

## **APP\_ENV**

The current environment, such as 'Development', 'Staging' or 'Production'

## **API\_ENDPOINT**

The URL of the API endpoint to use in the current environment

## **API\_KEY**

The API key to use for requests to the API

# Testing in Production

A test suite should be written to test functionality of your application. It's important to also have a few tests that run in production. Tests should cover the most important parts of your application since the change to production and the real environment can uncover new bugs. Consider an A/B testing framework.

## Test Suite

Comprehensive test suite covering the application functionality

## Tests in Production

A handful of important tests that run in production

## A/B Testing

Testing framework to ensure the deployment's stability and performance in different environments

# Continuous Integration and Deployment

A CI/CD pipeline allows you to automate the testing and deployment processes, delivering new features and updates faster and more reliably. It integrates code from various developers to reduce bugs, test for unit tests and regression tests. Consider using a tool like CircleCI or TravisCI along with deployment services like AWS or Heroku.

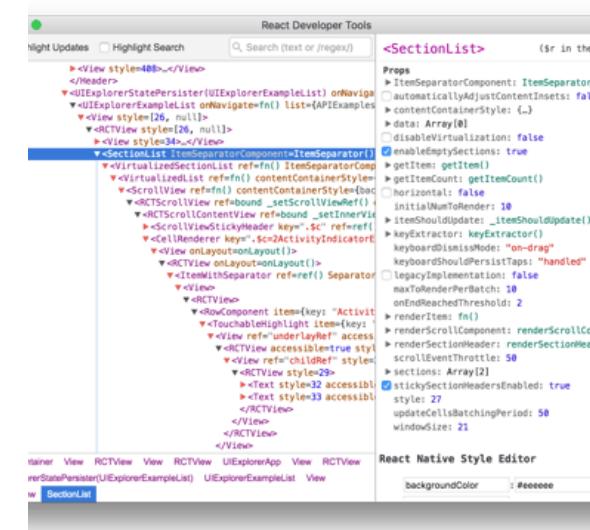
Continuous integration is like sweeping the floors. You do it every day so that nobody notices. When there's no continuous integration, everybody notices.

# Performance Monitoring

Performance monitoring is a crucial part of maintaining a modern ReactJS app. It allows you to measure and analyze the performance of your application in real-time, fix errors, and improve the user experience. Consider using tools like New Relic or Datadog for monitoring your app's performance, along with React profiling and debugging tools like React Developer Tools. Ensure your app can handle scale without buckling under load.



Performance dashboard in Grafana



Measure React rendering performance using React Developer Tools

# Security Considerations

Security is a vital consideration when deploying your ReactJS app. You need to ensure the app is safe by securing it against XSS attacks, adding authentication and authorization on the frontend and backend, and using SSL to encrypt the data in transit. Also ensure developers have a strong access control protocols and conducting regular penetration testing.

## XSS Attacks

Secure your React app against Cross-Site Scripting attacks such as protection against injection of scripts.

## Authentication and Authorization

Secure user's online identities by requiring authentication and authorizing task sequences.

## SSL

Secure Sockets Layer (SSL) encrypts the data flowing between your website and your users. This prevents attackers from intercepting sensitive data flowing between the two.

# Conclusion

In this document, we covered critical considerations when deploying your ReactJS app to production. These include environment variables, performance monitoring, security, deploying approaches, continuous integration, and testing in production. By adhering to these best practices, you'll be able to deploy your ReactJS app securely, efficiently, and with ease. Keep this document as your checklist and good luck with your deployments!

# API Integration and Asynchronous Operations in Reactjs

Become proficient in integrating APIs and performing asynchronous operations in React. Gain valuable insight into React state and lifecycle methods to manage data and optimize performance. In this guide, we'll cover all you need to know to master the techniques of API integration in React.



by Curious Coder ([CodeWithCurious.com](http://CodeWithCurious.com))

# What is an API Integration?

## Definition

An API integration is a way to connect disparate systems to share data and functionality. APIs are used to send and receive data from a server. Integrations can help automate tasks and improve efficiency.

## Why Integrate?

API integration helps businesses connect systems and automate tasks, which saves time and reduces errors. Integrating APIs also means businesses can access more functionality than standalone systems.

# Why use Asynchronous Operations?

Asynchronous Operations are a fundamental concept in modern web development. They help ensure that pages load quickly, responding to user-triggered events in near real-time, giving end users the seamless experience they expect.

In this section, we'll explore why asynchronous operations are so important when working with APIs and how React manages them.

# React state and lifecycle methods

State	Lifecycle Methods
State is an object that represents the internal state of a component	ComponentDidMount is called once when the component is first mounted.
State can change in response to user events or API calls	ShouldComponentUpdate lets us control if the component should re-render or not, optimizing performance.
State can be passed down to child components via props	ComponentWillUnmount is called just before the component is unmounted and destroyed.

# Working with asynchronous data in React

## 1 Asynchronous Data Retrieval

Pull data from the API asynchronously without blocking UI updates.

## 2 Updating the State

Update the React state with data once it arrives from the API.

## 3 Handling Errors and Edge Cases

Handle errors that occur during the request and decide how best to proceed.

# Implementing API integration and asynchronous operations in React

## **Set Up the API Call**

Create a function to fetch the data from the API.

## **Set Up the React Component**

Create a new component to display the data.

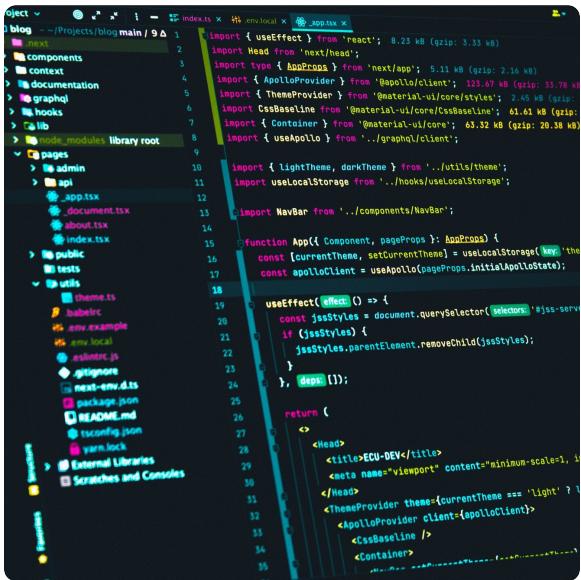
## **Update the State**

Update the state of each element with data from the API.

## **Render the Component**

Render the component to display the data to the user.

# Conclusion



```
import { useEffect } from 'react';
import Head from 'next/head';
import type { AppProps } from 'next/app';
import { ApolloProvider } from 'apollo/client';
import { ThemeProvider } from '@material-ui/core/styles';
import CssBaseline from '@material-ui/core/CssBaseline';
import { Container } from '@material-ui/core';
import { useApollo } from '../graphql/client';

import { lightTheme, darkTheme } from '../utils/theme';
import useLocalStorage from '../hooks/useLocalStorage';

import NavBar from '../components/NavBar';

function App({ Component, pageProps }: AppProps) {
  const [currentTheme, setCurrentTheme] = useLocalStorage('key', 'theme');
  const apolloClient = useApollo(pageProps.initialApolloState);

  useEffect(() => {
    const jsStyles = document.querySelector(`[data-theme=${currentTheme}]`);
    if (jsStyles) {
      jsStyles.parentElement.removeChild(jsStyles);
    }
  }, [currentTheme]);
}

return (
  <Head>
    <title>ECU-DEV</title>
    <meta name="viewport" content="minimum-scale=1, initial-scale=1, width=device-width, height=device-height, user-scalable=no" />
  </Head>
  <ThemeProvider theme={currentTheme === 'light' ? lightTheme : darkTheme}>
    <ApolloProvider client={apolloClient}>
      <CssBaseline />
      <Container>
        <Component {...pageProps} />
      </Container>
    </ApolloProvider>
  </ThemeProvider>
)
```

React is an excellent library for building dynamic web applications that rely on fast, asynchronous data retrieval.

Asynchronous data retrieval and API integration are critical tools for building high-quality, modern web applications. By implementing these tools in React, developers can deliver faster, more responsive web experiences to their users.



By integrating with existing APIs, developers can save time and build more advanced applications.

# Testing React Applications

In this article, we'll explore the different types of testing available for React applications. We'll discuss the tools, processes, and strategies you can use to ensure that your code is reliable and efficient. From unit testing to end-to-end testing, we'll cover it all. Let's get started!



by Curious Coder ([CodeWithCurious.com](https://CodeWithCurious.com))

# Setting up the Environment

Before we dive into the world of testing, we need to set up our environment. We'll need the right tools and configurations to make sure our tests run smoothly. We'll review the different options available for building and testing React applications, and we'll explore the benefits of each one.

## Jest

One of the most popular testing frameworks for React applications. It comes with a suite of testing features, including snapshot testing, mocking, and code coverage analysis.

## Enzyme

A lightweight testing framework that makes it easy to test React components. It provides a set of intuitive APIs that make component testing a breeze.

## Cypress

An end-to-end testing tool built for the modern web. It allows you to test the entire application, from the front end to the back end, using a single framework.

# Unit Testing

Unit testing helps us ensure that individual units of code work as expected. In this section, we'll explore the basics of unit testing and how it applies to React applications. We'll also discuss different strategies for testing components and Redux, ensuring that each piece of code is reliable and efficient.

Unit testing is a critical part of building reliable software. It allows us to catch bugs early and ensures that each piece of code works as intended.

# Integration Testing

Integration testing takes unit testing to the next level. It allows us to test interactions between components and Redux and ensure that our entire application works as expected. In this section, we'll explore the differences between unit and integration testing and how to approach integration testing in your React applications.



Collaboration is key when it comes to integration testing. Make sure your team is working together to test every aspect of the application.

# **End-to-End Testing**

End-to-end testing allows us to test the entire application, from start to finish. It ensures that each part of the application works together seamlessly and that all features are functioning as intended. In this section, we'll explore the basics of end-to-end testing and how it applies to React applications. We'll also discuss different testing frameworks available and which one is best for your team.

# Choosing Testing Tools

There are many tools available for testing React applications, each with its own strengths and weaknesses. In this section, we'll explore the different tools available and help you choose the right tools for your team and your application. We'll discuss the benefits and drawbacks of each tool, and we'll provide tips for integrating them into your workflow.

## Jest

The go-to testing framework for React applications. Offers easy configuration and a powerful suite of testing features.

## Cypress

The end-to-end testing tool of choice for many. It offers a powerful set of features for testing the entire application.

## Enzyme

A lightweight testing framework that makes it easy to test React components. It provides a suite of intuitive APIs that make testing a breeze.

## React Testing Library

A simple and lightweight library for testing React components. It offers a set of intuitive APIs for testing and makes it easy to write clear and concise tests.

# Testing Best Practices

There are many best practices to keep in mind when testing React applications. In this section, we'll cover some of the most important ones. From writing clear and concise tests to ensuring that your tests are reliable and efficient, we'll help you create a solid testing strategy for your team.

Quality code requires quality testing. Make sure you're taking the time to test your code thoroughly.

# Conclusion

Testing React applications is an essential part of the development process. It ensures that your code is reliable and efficient and catches bugs early on. In this article, we've explored the different types of testing available and provided tips and best practices for implementing a solid testing strategy. With these tools and strategies in hand, you can create a reliable and efficient React application that meets the needs of your users.