Cheat Sheet for comprehensive React

React Basics

Components

- **Functional Components**:

```
function MyComponent() {
  return <div>Hello, World!</div>;
}
```

- **Class Components**:

```
class MyComponent extends React.Component {
  render() {
    return <div>Hello, World!</div>;
  }
}
```

JSX

- **Basic Syntax**:

```
const element = <h1>Hello, {name}</h1>;
```

- **Embedding Expressions**:

```
const name = "React";
const element = <h1>Hello, {name}</h1>;
```

- **Attributes**:

```
const element = <img src={user.avatarUrl} alt="User Avatar" />;
```

State and Props

State

- **Initializing State**:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }
}
```

- **Updating State**:

```
this.setState({ count: this.state.count + 1 });
```

Props

- **Passing Props**:

```
<MyComponent name="React" />
```

- **Accessing Props**:

```
function MyComponent(props) {
  return <div>Hello, {props.name}</div>;
}
```

Event Handling

Synthetic Events

- **Basic Event Handling**:

```
function MyComponent() {
  function handleClick() {
    alert('Button clicked');
  }
  return <button onClick={handleClick}>Click me</button>;
}
```

- **Event Binding**:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
```

```
  handleClick() {
    alert('Button clicked');
  }
  render() {
    return <button onClick={this.handleClick}>Click me</button>;
  }
}
```

Conditional Rendering

If-Else

- **Inline If with Logical && Operator**:

```
{isLoggedIn && <h1>Welcome back!</h1>}
```

- **Inline If-Else with Conditional Operator**:

```
{isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please log in</h1>}
```

Lists and Keys

Rendering Lists

- **Basic List Rendering**:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>{number}</li>
);
```

- **Using Keys**:

```
const listItems = numbers.map((number) =>
  <li key={number.toString()}>{number}</li>
);
```

Hooks

useState

- **Basic Usage**:

```
import React, { useState } from 'react';

function MyComponent() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useEffect

- **Basic Usage**:

```
import React, { useState, useEffect } from 'react';

function MyComponent() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Context API

Creating Context

- **Basic Usage**:

```
const MyContext = React.createContext(defaultValue);
```

Providing Context

- **Basic Usage**:

```
<MyContext.Provider value={/* some value */}>
  <MyComponent />
</MyContext.Provider>
```

Consuming Context

- **Basic Usage**:

```
function MyComponent() {
  const value = useContext(MyContext);
  return <div>{value}</div>;
}
```

Routing

Basic Routing

- **Using react-router-dom**:

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

function App() {
  return (
    <Router>
      <Switch>
        <Route path="/about">
          <About />
        </Route>
        <Route path="/">
          <Home />
        </Route>
      </Switch>
    </Router>
  );
}
```

Performance Optimization

React.memo

- **Basic Usage**:

```
const MyComponent = React.memo(function MyComponent(props) {
  /* render using props */
});
```

useCallback

- **Basic Usage**:

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

useMemo

- **Basic Usage**:

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a,
b]);
```

Forms

Controlled Components

- **Basic Usage**:

```
function MyForm() {
  const [value, setValue] = useState('');

  const handleChange = (event) => {
    setValue(event.target.value);
  };

  const handleSubmit = (event) => {
    alert('A name was submitted: ' + value);
    event.preventDefault();
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={value} onChange={handleChange} />
      </label>
      <input type="submit" value="Submit" />
```

```
      </form>
    );
  }
```

Error Boundaries

Basic Usage

- **Creating an Error Boundary**:

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    logErrorToMyService(error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

Advanced Topics

Portals

- **Basic Usage**:

```
ReactDOM.createPortal(child, container);
```

Refs

- **Creating Refs**:

```
class MyComponent extends React.Component {
  constructor(props) {
```

```
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

Fragments

- **Basic Usage**:

```
function MyComponent() {
  return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  );
}
```

Tips and Tricks

Code Splitting

- **Using React.lazy**:

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));
```

Error Handling in Promises

- **Using try-catch with async/await**:

```
async function fetchData() {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    return data;
  } catch (error) {
    console.error('Error fetching data:', error);
  }
}
```

Debugging

- **Using React Developer Tools**:

- Install the React Developer Tools extension for Chrome or Firefox.
- Use the Components and Profiler tabs to inspect and profile your React components.

Best Practices

Component Naming

- Use PascalCase for component names.
- Example: `MyComponent.js`

File Structure

- Organize files by feature or component.
- Example:

```
src/
├── components/
│   ├── Header/
│   │   ├── Header.js
│   │   ├── Header.css
│   ├── Footer/
│   │   ├── Footer.js
│   │   ├── Footer.css
├── pages/
│   ├── HomePage.js
│   ├── AboutPage.js
```

Code Formatting

- Use Prettier for consistent code formatting.
- Example:

```
{
  "singleQuote": true,
  "trailingComma": "all",
  "printWidth": 80
}
```

Additional Resources

Official Documentation

- [React Documentation](https://reactjs.org/docs/getting-started.html)

Community Resources

- [Reactiflux Discord](https://www.reactiflux.com/)
- [Stack Overflow](https://stackoverflow.com/questions/tagged/reactjs)

Learning Platforms

- [Codecademy](https://www.codecademy.com/learn/react-101)
- [Udemy](https://www.udemy.com/topic/react/)

Conclusion

This cheat sheet covers the essential aspects of React, from basic concepts to advanced topics. Use it as a quick reference to build and optimize your React applications.

By Ahmed Baheeg Khorshid

ver 1.0