Complete React.js Cheat Sheet - Beginner to Advanced

Table of Contents

Getting Started

Installation

```
# Create React App
npx create-react-app my-app
cd my-app
npm start

# With Vite (faster alternative)
npm create vite@latest my-app -- --template react
cd my-app
npm install
npm run dev

# With TypeScript
npx create-react-app my-app --template typescript
```

Basic App Structure

```
// App.js
import React from "react";
import "./App.css";

function App() {
  return (
    <div className="App">
      <h1>Hello, React!</h1>
    </div>
  );
}

export default App;
```

JSX Fundamentals

JSX Syntax Rules

```
// JSX must return a single parent element
function Component() {
  return (
    <div>
      <h1>Title</h1>
      <p>Content</p>
    </div>
  );
}

// Or use React Fragment
function Component() {
  return (
    <>
      <h1>Title</h1>
      <p>Content</p>
    </>
  );
}

// Or explicit Fragment
import { Fragment } from "react";
function Component() {
  return (
    <Fragment>
      <h1>Title</h1>
      <p>Content</p>
    </Fragment>
  );
}
```

JavaScript in JSX

```
function Greeting() {
  const name = "John";
  const age = 25;

  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>You are {age} years old</p>
      <p>Next year you'll be {age + 1}</p>
      <p>{age >= 18 ? "Adult" : "Minor"}</p>
    </div>
  );
}
```

JSX Attributes

```
function Example() {
  const imgSrc = "image.jpg";
  const isActive = true;

  return (
    <div>
      {/* className instead of class */}
      <div className={isActive ? "active" : "inactive"}>
        <img src={imgSrc} alt="Description" />
      </div>

      {/* camelCase for attributes */}
      <input type="text" onChange={handleChange} maxLength={50} autoFocus />

      {/* Style as object */}
      <div
        style={{
          backgroundColor: "blue",
          fontSize: "16px",
          marginTop: "10px",
        }}
      >
        Styled div
      </div>
    </div>
  );
}
```

Components

Function Components

```jsx
// Basic function component
function Welcome() {
  return <h1>Hello, World!</h1>;
}

// Arrow function component
const Welcome = () => {
  return <h1>Hello, World!</h1>;
};

// Implicit return
const Welcome = () => <h1>Hello, World!</h1>;

// With parameters (props)
const Welcome = (props) => <h1>Hello, {props.name}!</h1>;

// With destructuring
const Welcome = ({ name, age }) => (
  <div>
    <h1>Hello, {name}!</h1>
    <p>Age: {age}</p>
  </div>
);
```

Class Components (Legacy)

```jsx
import React, { Component } from "react";

class Welcome extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
  }

  render() {
    return (
      <div>
        <h1>Hello, {this.props.name}!</h1>
        <p>Count: {this.state.count}</p>
      </div>
    );
  }
}
```

Component Composition

```
function Header() {
  return <h1>My Website</h1>;
}

function Sidebar() {
  return <aside>Sidebar content</aside>;
}

function Main() {
  return <main>Main content</main>;
}

function Layout() {
  return (
    <div>
      <Header />
      <div className="container">
        <Sidebar />
        <Main />
      </div>
    </div>
  );
}
```

Props

Basic Props

```
// Parent component
function App() {
  return (
    <div>
      <Welcome name="Alice" age={30} />
      <Welcome name="Bob" age={25} />
    </div>
  );
}

// Child component
function Welcome(props) {
  return (
    <div>
      <h1>Hello, {props.name}!</h1>
      <p>Age: {props.age}</p>
    </div>
  );
}

// With destructuring
```

```
function Welcome({ name, age }) {
  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>Age: {age}</p>
    </div>
  );
}
```

Default Props

```
function Welcome({ name = "Guest", age = 0 }) {
  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>Age: {age}</p>
    </div>
  );
}

// Or using defaultProps (legacy)
Welcome.defaultProps = {
  name: "Guest",
  age: 0,
};
```

Props Children

```
function Card({ children, title }) {
  return (
    <div className="card">
      <h2>{title}</h2>
      <div className="card-content">{children}</div>
    </div>
  );
}

// Usage
function App() {
  return (
    <Card title="My Card">
      <p>This is inside the card</p>
      <button>Click me</button>
    </Card>
  );
}
```

Prop Types (for type checking)

```
import PropTypes from "prop-types";

function Welcome({ name, age, isActive }) {
  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>Age: {age}</p>
      <p>Status: {isActive ? "Active" : "Inactive"}</p>
    </div>
  );
}

Welcome.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number,
  isActive: PropTypes.bool,
};

Welcome.defaultProps = {
  age: 0,
  isActive: false,
};
```

State & useState Hook

Basic useState

```
import { useState } from "react";

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
}
```

Multiple State Variables

```
function UserProfile() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
```

```
  const [age, setAge] = useState(0);

  return (
    <form>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Name"
      />
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
      />
      <input
        type="number"
        value={age}
        onChange={(e) => setAge(parseInt(e.target.value))}
        placeholder="Age"
      />
    </form>
  );
}
```

Object State

```
function UserProfile() {
  const [user, setUser] = useState({
    name: "",
    email: "",
    age: 0,
  });

  const updateUser = (field, value) => {
    setUser((prevUser) => ({
      ...prevUser,
      [field]: value,
    }));
  };

  return (
    <form>
      <input
        type="text"
        value={user.name}
        onChange={(e) => updateUser("name", e.target.value)}
        placeholder="Name"
      />
      <input
```

```
        type="email"
        value={user.email}
        onChange={(e) => updateUser("email", e.target.value)}
        placeholder="Email"
      />
    </form>
  );
}
```

Array State

```
function TodoList() {
  const [todos, setTodos] = useState([]);
  const [input, setInput] = useState("");

  const addTodo = () => {
    if (input.trim()) {
      setTodos([
        ...todos,
        {
          id: Date.now(),
          text: input,
          completed: false,
        },
      ]);
      setInput("");
    }
  };

  const removeTodo = (id) => {
    setTodos(todos.filter((todo) => todo.id !== id));
  };

  const toggleTodo = (id) => {
    setTodos(
      todos.map((todo) =>
        todo.id === id ? { ...todo, completed: !todo.completed } : todo
      )
    );
  };

  return (
    <div>
      <input
        value={input}
        onChange={(e) => setInput(e.target.value)}
        onKeyPress={(e) => e.key === "Enter" && addTodo()}
      />
      <button onClick={addTodo}>Add Todo</button>

      <ul>
```

```
        {todos.map((todo) => (
          <li key={todo.id}>
            <span
              style={{
                textDecoration: todo.completed ? "line-through" : "none",
              }}
              onClick={() => toggleTodo(todo.id)}
            >
              {todo.text}
            </span>
            <button onClick={() => removeTodo(todo.id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Event Handling

Basic Event Handling

```
function EventExample() {
  const handleClick = () => {
    alert("Button clicked!");
  };

  const handleInputChange = (e) => {
    console.log("Input value:", e.target.value);
  };

  const handleSubmit = (e) => {
    e.preventDefault(); // Prevent default form submission
    console.log("Form submitted");
  };

  return (
    <div>
      <button onClick={handleClick}>Click me</button>

      <input onChange={handleInputChange} />

      <form onSubmit={handleSubmit}>
        <input type="text" />
        <button type="submit">Submit</button>
      </form>
    </div>
  );
}
```

Event Object Properties

```
function EventDetails() {
  const handleEvent = (e) => {
    console.log("Event type:", e.type);
    console.log("Target element:", e.target);
    console.log("Current target:", e.currentTarget);
    console.log("Key pressed:", e.key); // for keyboard events
    console.log("Mouse button:", e.button); // for mouse events
  };

  return (
    <div>
      <button onClick={handleEvent}>Click</button>
      <input onKeyDown={handleEvent} />
      <div onMouseEnter={handleEvent}>Hover me</div>
    </div>
  );
}
```

Passing Parameters to Event Handlers

```
function ParameterExample() {
  const handleClick = (message, e) => {
    console.log(message);
    console.log("Event:", e);
  };

  const items = ["Apple", "Banana", "Cherry"];

  return (
    <div>
      {/* Method 1: Arrow function */}
      <button onClick={(e) => handleClick("Hello!", e)}>Click me</button>

      {/* Method 2: In a list */}
      {items.map((item, index) => (
        <button key={index} onClick={() => console.log(`Clicked ${item}`)}>
          {item}
        </button>
      ))}
    </div>
  );
}
```

Conditional Rendering

If-Else with &&

```
function ConditionalExample({ isLoggedIn, user }) {
  return (
    <div>
      {isLoggedIn && <h1>Welcome back, {user.name}!</h1>}
      {!isLoggedIn && <h1>Please log in</h1>}
    </div>
  );
}
```

Ternary Operator

```
function LoginStatus({ isLoggedIn, user }) {
  return (
    <div>
      <h1>{isLoggedIn ? `Welcome, ${user.name}!` : "Please log in"}</h1>

      <button>{isLoggedIn ? "Logout" : "Login"}</button>
    </div>
  );
}
```

Switch-like Conditional Rendering

```
function StatusMessage({ status }) {
  const getStatusMessage = () => {
    switch (status) {
      case "loading":
        return <div>Loading...</div>;
      case "success":
        return <div>Success!</div>;
      case "error":
        return <div>Something went wrong</div>;
      default:
        return <div>Unknown status</div>;
    }
  };

  return (
    <div>
      <h1>Status</h1>
      {getStatusMessage()}
    </div>
  );
}
```

Conditional Styling

```
function ConditionalStyling({ isActive, isError }) {
  return (
    <div
      className={`
        button
        ${isActive ? "active" : ""}
        ${isError ? "error" : ""}
      `.trim()}
      style={{
        backgroundColor: isError ? "red" : isActive ? "green" : "gray",
        opacity: isActive ? 1 : 0.5,
      }}
    >
      Button
    </div>
  );
}
```

Lists & Keys

Basic List Rendering

```
function ItemList() {
  const items = ["Apple", "Banana", "Cherry", "Date"];

  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

Complex List with Objects

```
function UserList() {
  const users = [
    { id: 1, name: "Alice", email: "alice@example.com" },
    { id: 2, name: "Bob", email: "bob@example.com" },
    { id: 3, name: "Charlie", email: "charlie@example.com" },
  ];

  return (
    <div>
      {users.map((user) => (
        <div key={user.id} className="user-card">
```

```
            <h3>{user.name}</h3>
            <p>{user.email}</p>
          </div>
        ))}
      </div>
    );
  }
```

Filtered Lists

```
function FilteredList() {
  const [filter, setFilter] = useState("");
  const items = ["Apple", "Banana", "Cherry", "Date", "Elderberry"];

  const filteredItems = items.filter((item) =>
    item.toLowerCase().includes(filter.toLowerCase())
  );

  return (
    <div>
      <input
        type="text"
        value={filter}
        onChange={(e) => setFilter(e.target.value)}
        placeholder="Filter items..."
      />

      <ul>
        {filteredItems.map((item, index) => (
          <li key={item}>{item}</li>
        ))}
      </ul>
    </div>
  );
}
```

Keys Best Practices

```
// ❌ Bad: Using array index as key
function BadList({ items }) {
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item.name}</li>
      ))}
    </ul>
  );
}
```

```
// ☑ Good: Using unique identifier as key
function GoodList({ items }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
}

// ☑ Good: Using stable content as key (if no unique id)
function ContentKeyList({ items }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.name}>{item.name}</li>
      ))}
    </ul>
  );
}
```

Forms

Controlled Components

```
function ContactForm() {
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    message: "",
    category: "general",
    subscribe: false,
  });

  const handleInputChange = (e) => {
    const { name, value, type, checked } = e.target;
    setFormData((prev) => ({
      ...prev,
      [name]: type === "checkbox" ? checked : value,
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Form submitted:", formData);
    // Handle form submission
  };

  return (
```

```
      <form onSubmit={handleSubmit}>
        <div>
          <label>
            Name:
            <input
              type="text"
              name="name"
              value={formData.name}
              onChange={handleInputChange}
              required
            />
          </label>
        </div>

        <div>
          <label>
            Email:
            <input
              type="email"
              name="email"
              value={formData.email}
              onChange={handleInputChange}
              required
            />
          </label>
        </div>

        <div>
          <label>
            Message:
            <textarea
              name="message"
              value={formData.message}
              onChange={handleInputChange}
              rows={4}
            />
          </label>
        </div>

        <div>
          <label>
            Category:
            <select
              name="category"
              value={formData.category}
              onChange={handleInputChange}
            >
              <option value="general">General</option>
              <option value="support">Support</option>
              <option value="feedback">Feedback</option>
            </select>
          </label>
        </div>
```

```
        <div>
          <label>
            <input
              type="checkbox"
              name="subscribe"
              checked={formData.subscribe}
              onChange={handleInputChange}
            />
            Subscribe to newsletter
          </label>
        </div>

        <button type="submit">Submit</button>
      </form>
    );
  }
```

Form Validation

```
function ValidatedForm() {
  const [formData, setFormData] = useState({
    email: "",
    password: "",
    confirmPassword: "",
  });
  const [errors, setErrors] = useState({});

  const validateForm = () => {
    const newErrors = {};

    // Email validation
    if (!formData.email) {
      newErrors.email = "Email is required";
    } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
      newErrors.email = "Email is invalid";
    }

    // Password validation
    if (!formData.password) {
      newErrors.password = "Password is required";
    } else if (formData.password.length < 6) {
      newErrors.password = "Password must be at least 6 characters";
    }

    // Confirm password validation
    if (formData.password !== formData.confirmPassword) {
      newErrors.confirmPassword = "Passwords do not match";
    }

    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
```

```
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (validateForm()) {
      console.log("Form is valid:", formData);
    }
  };

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prev) => ({
      ...prev,
      [name]: value,
    }));

    // Clear error when user starts typing
    if (errors[name]) {
      setErrors((prev) => ({
        ...prev,
        [name]: "",
      }));
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <input
          type="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
          placeholder="Email"
        />
        {errors.email && <span className="error">{errors.email}</span>}
      </div>

      <div>
        <input
          type="password"
          name="password"
          value={formData.password}
          onChange={handleChange}
          placeholder="Password"
        />
        {errors.password && <span className="error">{errors.password}</span>}
      </div>

      <div>
        <input
          type="password"
          name="confirmPassword"
          value={formData.confirmPassword}
```

```
            onChange={handleChange}
            placeholder="Confirm Password"
          />
          {errors.confirmPassword && (
            <span className="error">{errors.confirmPassword}</span>
          )}
        </div>

        <button type="submit">Submit</button>
      </form>
    );
}
```

---

useEffect Hook

Basic useEffect

```
import { useState, useEffect } from "react";

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

  // Effect runs after every render
  useEffect(() => {
    document.title = `Count: ${count}`;
  });

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

useEffect with Dependencies

```
function EffectWithDependencies() {
  const [count, setCount] = useState(0);
  const [name, setName] = useState("");

  // Effect runs only when count changes
  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]);

  // Effect runs only once (component mount)
  useEffect(() => {
```

```
    console.log("Component mounted");
  }, []);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>

      <input
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Name"
      />
    </div>
  );
}
```

Cleanup with useEffect

```
function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setSeconds((prev) => prev + 1);
    }, 1000);

    // Cleanup function
    return () => {
      clearInterval(intervalId);
    };
  }, []); // Empty dependency array = runs once

  return <div>Timer: {seconds} seconds</div>;
}

function WindowSize() {
  const [windowSize, setWindowSize] = useState({
    width: window.innerWidth,
    height: window.innerHeight,
  });

  useEffect(() => {
    const handleResize = () => {
      setWindowSize({
        width: window.innerWidth,
        height: window.innerHeight,
      });
    };

    window.addEventListener("resize", handleResize);
```

```
    // Cleanup
    return () => {
      window.removeEventListener("resize", handleResize);
    };
  }, []);

  return (
    <div>
      Window size: {windowSize.width} x {windowSize.height}
    </div>
  );
}
```

Data Fetching with useEffect

```
function UserProfile({ userId }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    let isCancelled = false;

    const fetchUser = async () => {
      try {
        setLoading(true);
        setError(null);

        const response = await fetch(`/api/users/${userId}`);
        if (!response.ok) {
          throw new Error("User not found");
        }

        const userData = await response.json();

        if (!isCancelled) {
          setUser(userData);
        }
      } catch (err) {
        if (!isCancelled) {
          setError(err.message);
        }
      } finally {
        if (!isCancelled) {
          setLoading(false);
        }
      }
    };

    fetchUser();
```

```
    // Cleanup to prevent setting state on unmounted component
    return () => {
      isCancelled = true;
    };
  }, [userId]);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <div>
      <h2>{user.name}</h2>
      <p>{user.email}</p>
    </div>
  );
}
```

---

useContext Hook

Creating and Using Context

```
import { createContext, useContext, useState } from "react";

// Create context
const ThemeContext = createContext();

// Context provider component
function ThemeProvider({ children }) {
  const [theme, setTheme] = useState("light");

  const toggleTheme = () => {
    setTheme((prev) => (prev === "light" ? "dark" : "light"));
  };

  const value = {
    theme,
    toggleTheme,
  };

  return (
    <ThemeContext.Provider value={value}>{children}</ThemeContext.Provider>
  );
}

// Custom hook to use theme context
function useTheme() {
  const context = useContext(ThemeContext);
  if (!context) {
    throw new Error("useTheme must be used within ThemeProvider");
```

```
  }
  return context;
}

// Component using context
function Header() {
  const { theme, toggleTheme } = useTheme();

  return (
    <header
      style={{
        backgroundColor: theme === "light" ? "#fff" : "#333",
        color: theme === "light" ? "#333" : "#fff",
      }}
    >
      <h1>My App</h1>
      <button onClick={toggleTheme}>
        Switch to {theme === "light" ? "dark" : "light"} theme
      </button>
    </header>
  );
}

// App component
function App() {
  return (
    <ThemeProvider>
      <Header />
      {/* Other components */}
    </ThemeProvider>
  );
}
```

User Authentication Context

```
const AuthContext = createContext();

function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Check if user is logged in
    const token = localStorage.getItem("token");
    if (token) {
      // Validate token and get user data
      fetchUserProfile(token);
    } else {
      setLoading(false);
    }
  }, []);
```

```
  const login = async (email, password) => {
    try {
      const response = await fetch("/api/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password }),
      });

      const data = await response.json();

      if (response.ok) {
        localStorage.setItem("token", data.token);
        setUser(data.user);
        return { success: true };
      } else {
        return { success: false, error: data.message };
      }
    } catch (error) {
      return { success: false, error: "Network error" };
    }
  };

  const logout = () => {
    localStorage.removeItem("token");
    setUser(null);
  };

  const value = {
    user,
    login,
    logout,
    loading,
  };

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
}

function useAuth() {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error("useAuth must be used within AuthProvider");
  }
  return context;
}
```

---

useReducer Hook

Basic useReducer

```jsx
import { useReducer } from "react";

// Reducer function
function counterReducer(state, action) {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    case "RESET":
      return { count: 0 };
    case "SET":
      return { count: action.payload };
    default:
      throw new Error(`Unknown action type: ${action.type}`);
  }
}

function Counter() {
  const [state, dispatch] = useReducer(counterReducer, { count: 0 });

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: "INCREMENT" })}>Increment</button>
      <button onClick={() => dispatch({ type: "DECREMENT" })}>Decrement</button>
      <button onClick={() => dispatch({ type: "RESET" })}>Reset</button>
      <button onClick={() => dispatch({ type: "SET", payload: 10 })}>
        Set to 10
      </button>
    </div>
  );
}
```

Complex State with useReducer

```jsx
// Todo list reducer
function todoReducer(state, action) {
  switch (action.type) {
    case "ADD_TODO":
      return {
        ...state,
        todos: [
          ...state.todos,
          {
            id: Date.now(),
            text: action.payload,
            completed: false,
          },
        ],
```

```
      };
    case "TOGGLE_TODO":
      return {
        ...state,
        todos: state.todos.map((todo) =>
          todo.id === action.payload
            ? { ...todo, completed: !todo.completed }
            : todo
        ),
      };
    case "DELETE_TODO":
      return {
        ...state,
        todos: state.todos.filter((todo) => todo.id !== action.payload),
      };
    case "SET_FILTER":
      return {
        ...state,
        filter: action.payload,
      };
    case "CLEAR_COMPLETED":
      return {
        ...state,
        todos: state.todos.filter((todo) => !todo.completed),
      };
    default:
      throw new Error(`Unknown action type: ${action.type}`);
  }
}

function TodoApp() {
  const initialState = {
    todos: [],
    filter: "all", // 'all', 'active', 'completed'
  };

  const [state, dispatch] = useReducer(todoReducer, initialState);
  const [input, setInput] = useState("");

  const addTodo = () => {
    if (input.trim()) {
      dispatch({ type: "ADD_TODO", payload: input });
      setInput("");
    }
  };

  const filteredTodos = state.todos.filter((todo) => {
    if (state.filter === "active") return !todo.completed;
    if (state.filter === "completed") return todo.completed;
    return true;
  });

  return (
    <div>
```

```
        <input
          value={input}
          onChange={(e) => setInput(e.target.value)}
          onKeyPress={(e) => e.key === "Enter" && addTodo()}
        />
        <button onClick={addTodo}>Add Todo</button>

        <div>
          <button
            onClick={() => dispatch({ type: "SET_FILTER", payload: "all" })}
            style={{ fontWeight: state.filter === "all" ? "bold" : "normal" }}
          >
            All
          </button>
          <button
            onClick={() => dispatch({ type: "SET_FILTER", payload: "active" })}
            style={{ fontWeight: state.filter === "active" ? "bold" : "normal" }}
          >
            Active
          </button>
          <button
            onClick={() => dispatch({ type: "SET_FILTER", payload: "completed" })}
            style={{
              fontWeight: state.filter === "completed" ? "bold" : "normal",
            }}
          >
            Completed
          </button>
        </div>

        <ul>
          {filteredTodos.map((todo) => (
            <li key={todo.id}>
              <span
                style={{
                  textDecoration: todo.completed ? "line-through" : "none",
                  cursor: "pointer",
                }}
                onClick={() =>
                  dispatch({ type: "TOGGLE_TODO", payload: todo.id })
                }
              >
                {todo.text}
              </span>
              <button
                onClick={() =>
                  dispatch({ type: "DELETE_TODO", payload: todo.id })
                }
              >
                Delete
              </button>
            </li>
          ))}
        </ul>
```

```
      <button onClick={() => dispatch({ type: "CLEAR_COMPLETED" })}>
        Clear Completed
      </button>
    </div>
  );
}
```

Custom Hooks

Basic Custom Hook

```
// Custom hook for local storage
function useLocalStorage(key, initialValue) {
  const [storedValue, setStoredValue] = useState(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      console.error("Error reading from localStorage:", error);
      return initialValue;
    }
  });

  const setValue = (value) => {
    try {
      const valueToStore =
        value instanceof Function ? value(storedValue) : value;
      setStoredValue(valueToStore);
      window.localStorage.setItem(key, JSON.stringify(valueToStore));
    } catch (error) {
      console.error("Error writing to localStorage:", error);
    }
  };

  return [storedValue, setValue];
}

// Usage
function Settings() {
  const [theme, setTheme] = useLocalStorage("theme", "light");
  const [language, setLanguage] = useLocalStorage("language", "en");

  return (
    <div>
      <button onClick={() => setTheme(theme === "light" ? "dark" : "light")}>
        Theme: {theme}
      </button>
      <select value={language} onChange={(e) => setLanguage(e.target.value)}>
        <option value="en">English</option>
```

```
          <option value="es">Spanish</option>
          <option value="fr">French</option>
        </select>
      </div>
    );
  }
```

Fetch Data Hook

```
function useFetch(url, options = {}) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    let isCancelled = false;

    const fetchData = async () => {
      try {
        setLoading(true);
        setError(null);

        const response = await fetch(url, options);
        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }

        const result = await response.json();

        if (!isCancelled) {
          setData(result);
        }
      } catch (err) {
        if (!isCancelled) {
          setError(err.message);
        }
      } finally {
        if (!isCancelled) {
          setLoading(false);
        }
      }
    };

    fetchData();

    return () => {
      isCancelled = true;
    };
  }, [url, JSON.stringify(options)]);

  return { data, loading, error };
```

```
}

// Usage
function UserList() {
  const { data: users, loading, error } = useFetch("/api/users");

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

Form Hook

```
function useForm(initialValues, validate) {
  const [values, setValues] = useState(initialValues);
  const [errors, setErrors] = useState({});
  const [touched, setTouched] = useState({});

  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    const newValue = type === "checkbox" ? checked : value;

    setValues((prev) => ({
      ...prev,
      [name]: newValue,
    }));

    // Clear error when user starts typing
    if (errors[name]) {
      setErrors((prev) => ({
        ...prev,
        [name]: "",
      }));
    }
  };

  const handleBlur = (e) => {
    const { name } = e.target;
    setTouched((prev) => ({
      ...prev,
      [name]: true,
    }));

    if (validate) {
```

```
      const fieldErrors = validate(values);
      setErrors((prev) => ({
        ...prev,
        [name]: fieldErrors[name] || "",
      }));
    }
  };

  const handleSubmit = (onSubmit) => (e) => {
    e.preventDefault();

    if (validate) {
      const formErrors = validate(values);
      setErrors(formErrors);

      if (Object.keys(formErrors).length === 0) {
        onSubmit(values);
      }
    } else {
      onSubmit(values);
    }
  };

  const reset = () => {
    setValues(initialValues);
    setErrors({});
    setTouched({});
  };

  return {
    values,
    errors,
    touched,
    handleChange,
    handleBlur,
    handleSubmit,
    reset,
  };
}

// Usage
function LoginForm() {
  const validate = (values) => {
    const errors = {};

    if (!values.email) {
      errors.email = "Email is required";
    } else if (!/\S+@\S+\.\S+/.test(values.email)) {
      errors.email = "Email is invalid";
    }

    if (!values.password) {
      errors.password = "Password is required";
    } else if (values.password.length < 6) {
```

```
    errors.password = "Password must be at least 6 characters";
  }

  return errors;
};

const {
  values,
  errors,
  touched,
  handleChange,
  handleBlur,
  handleSubmit,
  reset,
} = useForm({ email: "", password: "" }, validate);

const onSubmit = (data) => {
  console.log("Form submitted:", data);
  reset();
};

return (
  <form onSubmit={handleSubmit(onSubmit)}>
    <div>
      <input
        type="email"
        name="email"
        value={values.email}
        onChange={handleChange}
        onBlur={handleBlur}
        placeholder="Email"
      />
      {touched.email && errors.email && (
        <span className="error">{errors.email}</span>
      )}
    </div>

    <div>
      <input
        type="password"
        name="password"
        value={values.password}
        onChange={handleChange}
        onBlur={handleBlur}
        placeholder="Password"
      />
      {touched.password && errors.password && (
        <span className="error">{errors.password}</span>
      )}
    </div>

    <button type="submit">Login</button>
  </form>
```

```
  );
}
```

Component Lifecycle

Functional Component Lifecycle with Hooks

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

  // ComponentDidMount equivalent
  useEffect(() => {
    console.log("Component mounted");

    // ComponentWillUnmount equivalent (cleanup)
    return () => {
      console.log("Component will unmount");
    };
  }, []);

  // ComponentDidUpdate equivalent
  useEffect(() => {
    console.log("Component updated, count:", count);
  }, [count]);

  // Combined mount and update
  useEffect(() => {
    console.log("Component mounted or count updated");
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

Class Component Lifecycle (Legacy)

```
class LifecycleClass extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
    console.log("Constructor");
  }

  static getDerivedStateFromProps(props, state) {
```

```
    console.log("getDerivedStateFromProps");
    return null; // No state update needed
  }

  componentDidMount() {
    console.log("Component mounted");
  }

  shouldComponentUpdate(nextProps, nextState) {
    console.log("shouldComponentUpdate");
    return true; // Always update
  }

  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log("getSnapshotBeforeUpdate");
    return null;
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    console.log("Component updated");
  }

  componentWillUnmount() {
    console.log("Component will unmount");
  }

  render() {
    console.log("Render");
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Increment
        </button>
      </div>
    );
  }
}
```

Error Boundaries

Class-based Error Boundary

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

  static getDerivedStateFromError(error) {
```

```
    // Update state to show error UI
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // Log error details
    console.error("Error caught by boundary:", error, errorInfo);
    this.setState({
      error: error,
      errorInfo: errorInfo,
    });
  }

  render() {
    if (this.state.hasError) {
      return (
        <div style={{ padding: "20px", border: "1px solid red" }}>
          <h2>Something went wrong.</h2>
          <details style={{ whiteSpace: "pre-wrap" }}>
            <summary>Error details</summary>
            {this.state.error && this.state.error.toString()}
            <br />
            {this.state.errorInfo.componentStack}
          </details>
          <button
            onClick={() =>
              this.setState({ hasError: false, error: null, errorInfo: null })
            }
          >
            Try again
          </button>
        </div>
      );
    }

    return this.props.children;
  }
}

// Usage
function App() {
  return (
    <ErrorBoundary>
      <Header />
      <Main />
      <Footer />
    </ErrorBoundary>
  );
}
```

Hook-based Error Boundary (with react-error-boundary)

```
// Install: npm install react-error-boundary
import { ErrorBoundary } from "react-error-boundary";

function ErrorFallback({ error, resetErrorBoundary }) {
  return (
    <div role="alert" style={{ padding: "20px", border: "1px solid red" }}>
      <h2>Something went wrong:</h2>
      <pre>{error.message}</pre>
      <button onClick={resetErrorBoundary}>Try again</button>
    </div>
  );
}

function MyErrorBoundary({ children }) {
  return (
    <ErrorBoundary
      FallbackComponent={ErrorFallback}
      onError={(error, errorInfo) => {
        console.error("Error logged:", error, errorInfo);
        // Send to error reporting service
      }}
      onReset={() => {
        // Reset app state if needed
      }}
    >
      {children}
    </ErrorBoundary>
  );
}
```

Performance Optimization

React.memo

```
// Memoize component to prevent unnecessary re-renders
const ExpensiveComponent = React.memo(function ExpensiveComponent({
  name,
  value,
}) {
  console.log("ExpensiveComponent rendered");

  // Expensive calculation
  const expensiveValue = useMemo(() => {
    return Array.from({ length: 1000000 }, (_, i) => i).reduce(
      (a, b) => a + b,
      0
    );
  }, []);
```

```
  return (
    <div>
      <h3>{name}</h3>
      <p>Value: {value}</p>
      <p>Expensive calculation: {expensiveValue}</p>
    </div>
  );
});

// With custom comparison function
const CustomMemoComponent = React.memo(
  function CustomMemoComponent({ user, posts }) {
    return (
      <div>
        <h3>{user.name}</h3>
        <p>Posts: {posts.length}</p>
      </div>
    );
  },
  (prevProps, nextProps) => {
    // Custom comparison logic
    return (
      prevProps.user.id === nextProps.user.id &&
      prevProps.posts.length === nextProps.posts.length
    );
  }
);
```

useMemo Hook

```
function ExpensiveCalculation({ items, multiplier }) {
  // Memoize expensive calculation
  const expensiveValue = useMemo(() => {
    console.log("Calculating expensive value...");
    return items.reduce((sum, item) => sum + item.value, 0) * multiplier;
  }, [items, multiplier]);

  // Memoize filtered items
  const filteredItems = useMemo(() => {
    console.log("Filtering items...");
    return items.filter((item) => item.active);
  }, [items]);

  return (
    <div>
      <p>Total value: {expensiveValue}</p>
      <p>Active items: {filteredItems.length}</p>
    </div>
  );
}
```

useCallback Hook

```
function TodoList({ todos }) {
  const [filter, setFilter] = useState("all");

  // Memoize callback to prevent child re-renders
  const handleToggle = useCallback((id) => {
    setTodos((prevTodos) =>
      prevTodos.map((todo) =>
        todo.id === id ? { ...todo, completed: !todo.completed } : todo
      )
    );
  }, []);

  const handleDelete = useCallback((id) => {
    setTodos((prevTodos) => prevTodos.filter((todo) => todo.id !== id));
  }, []);

  const filteredTodos = useMemo(() => {
    return todos.filter((todo) => {
      if (filter === "active") return !todo.completed;
      if (filter === "completed") return todo.completed;
      return true;
    });
  }, [todos, filter]);

  return (
    <div>
      <FilterButtons filter={filter} onFilterChange={setFilter} />
      {filteredTodos.map((todo) => (
        <TodoItem
          key={todo.id}
          todo={todo}
          onToggle={handleToggle}
          onDelete={handleDelete}
        />
      ))}
    </div>
  );
}

const TodoItem = React.memo(({ todo, onToggle, onDelete }) => {
  console.log(`TodoItem ${todo.id} rendered`);

  return (
    <div>
      <span
        onClick={() => onToggle(todo.id)}
        style={{ textDecoration: todo.completed ? "line-through" : "none" }}
      >
        {todo.text}
      </span>
```

```
      <button onClick={() => onDelete(todo.id)}>Delete</button>
    </div>
  );
});
```

Lazy Loading

```
import { lazy, Suspense } from "react";

// Lazy load components
const LazyComponent = lazy(() => import("./LazyComponent"));
const Dashboard = lazy(() => import("./Dashboard"));
const Profile = lazy(() => import("./Profile"));

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />
      </Suspense>

      <Suspense fallback={<div>Loading dashboard...</div>}>
        <Dashboard />
      </Suspense>
    </div>
  );
}

// Lazy loading with error boundary
function LazyWithErrorBoundary() {
  return (
    <ErrorBoundary>
      <Suspense fallback={<div>Loading...</div>}>
        <Profile />
      </Suspense>
    </ErrorBoundary>
  );
}
```

Advanced Patterns

Higher-Order Components (HOC)

```
// HOC for authentication
function withAuth(WrappedComponent) {
  return function AuthenticatedComponent(props) {
    const { user, loading } = useAuth();
```

```
    if (loading) {
      return <div>Loading...</div>;
    }

    if (!user) {
      return <div>Please log in to access this page.</div>;
    }

    return <WrappedComponent {...props} user={user} />;
  };
}

// Usage
const ProtectedDashboard = withAuth(Dashboard);

// HOC for loading state
function withLoading(WrappedComponent) {
  return function LoadingComponent({ isLoading, ...props }) {
    if (isLoading) {
      return <div>Loading...</div>;
    }

    return <WrappedComponent {...props} />;
  };
}

const UserListWithLoading = withLoading(UserList);
```

Render Props Pattern

```
// Mouse tracker with render props
function MouseTracker({ render }) {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useEffect(() => {
    const handleMouseMove = (e) => {
      setPosition({ x: e.clientX, y: e.clientY });
    };

    document.addEventListener("mousemove", handleMouseMove);

    return () => {
      document.removeEventListener("mousemove", handleMouseMove);
    };
  }, []);

  return render(position);
}

// Usage
function App() {
```

```
  return (
    <div>
      <MouseTracker
        render={({ x, y }) => (
          <div>
            Mouse position: {x}, {y}
          </div>
        )}
      />

      <MouseTracker
        render={({ x, y }) => (
          <div
            style={{
              position: "absolute",
              left: x,
              top: y,
              width: 10,
              height: 10,
              backgroundColor: "red",
              borderRadius: "50%",
            }}
          />
        )}
      />
    </div>
  );
}
```

Compound Components Pattern

```
// Tabs compound component
const TabsContext = createContext();

function Tabs({ children, defaultTab = 0 }) {
  const [activeTab, setActiveTab] = useState(defaultTab);

  return (
    <TabsContext.Provider value={{ activeTab, setActiveTab }}>
      <div className="tabs">{children}</div>
    </TabsContext.Provider>
  );
}

function TabList({ children }) {
  return <div className="tab-list">{children}</div>;
}

function Tab({ index, children }) {
  const { activeTab, setActiveTab } = useContext(TabsContext);
```

```
    return (
      <button
        className={`tab ${activeTab === index ? "active" : ""}`}
        onClick={() => setActiveTab(index)}
      >
        {children}
      </button>
    );
  }

  function TabPanels({ children }) {
    return <div className="tab-panels">{children}</div>;
  }

  function TabPanel({ index, children }) {
    const { activeTab } = useContext(TabsContext);

    if (activeTab !== index) return null;

    return <div className="tab-panel">{children}</div>;
  }

  // Attach components to main component
  Tabs.List = TabList;
  Tabs.Tab = Tab;
  Tabs.Panels = TabPanels;
  Tabs.Panel = TabPanel;

  // Usage
  function App() {
    return (
      <Tabs defaultTab={0}>
        <Tabs.List>
          <Tabs.Tab index={0}>Tab 1</Tabs.Tab>
          <Tabs.Tab index={1}>Tab 2</Tabs.Tab>
          <Tabs.Tab index={2}>Tab 3</Tabs.Tab>
        </Tabs.List>

        <Tabs.Panels>
          <Tabs.Panel index={0}>Content for Tab 1</Tabs.Panel>
          <Tabs.Panel index={1}>Content for Tab 2</Tabs.Panel>
          <Tabs.Panel index={2}>Content for Tab 3</Tabs.Panel>
        </Tabs.Panels>
      </Tabs>
    );
  }
```

Portal Pattern

```
import { createPortal } from "react-dom";
```

```
function Modal({ isOpen, onClose, children }) {
  if (!isOpen) return null;

  return createPortal(
    <div className="modal-overlay" onClick={onClose}>
      <div className="modal-content" onClick={(e) => e.stopPropagation()}>
        <button className="modal-close" onClick={onClose}>
          ×
        </button>
        {children}
      </div>
    </div>,
    document.body
  );
}

// Usage
function App() {
  const [isModalOpen, setIsModalOpen] = useState(false);

  return (
    <div>
      <button onClick={() => setIsModalOpen(true)}>Open Modal</button>

      <Modal isOpen={isModalOpen} onClose={() => setIsModalOpen(false)}>
        <h2>Modal Title</h2>
        <p>Modal content goes here</p>
      </Modal>
    </div>
  );
}
```

Testing

Jest and React Testing Library

```
// Component to test
function Counter({ initialCount = 0 }) {
  const [count, setCount] = useState(initialCount);

  return (
    <div>
      <span data-testid="count">Count: {count}</span>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
}
```

```
// Test file: Counter.test.js
import { render, screen, fireEvent } from "@testing-library/react";
import "@testing-library/jest-dom";
import Counter from "./Counter";

describe("Counter Component", () => {
  test("renders initial count", () => {
    render(<Counter initialCount={5} />);
    expect(screen.getByTestId("count")).toHaveTextContent("Count: 5");
  });

  test("increments count when increment button is clicked", () => {
    render(<Counter />);
    const incrementButton = screen.getByText("Increment");

    fireEvent.click(incrementButton);
    expect(screen.getByTestId("count")).toHaveTextContent("Count: 1");
  });

  test("decrements count when decrement button is clicked", () => {
    render(<Counter initialCount={5} />);
    const decrementButton = screen.getByText("Decrement");

    fireEvent.click(decrementButton);
    expect(screen.getByTestId("count")).toHaveTextContent("Count: 4");
  });

  test("resets count when reset button is clicked", () => {
    render(<Counter initialCount={10} />);
    const resetButton = screen.getByText("Reset");

    fireEvent.click(resetButton);
    expect(screen.getByTestId("count")).toHaveTextContent("Count: 0");
  });
});
```

Testing Hooks

```
// Custom hook to test
function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);
  const reset = () => setCount(initialValue);

  return { count, increment, decrement, reset };
}

// Test file: useCounter.test.js
import { renderHook, act } from "@testing-library/react";
```

```
import useCounter from "./useCounter";

describe("useCounter Hook", () => {
  test("should initialize with default value", () => {
    const { result } = renderHook(() => useCounter());
    expect(result.current.count).toBe(0);
  });

  test("should initialize with provided value", () => {
    const { result } = renderHook(() => useCounter(10));
    expect(result.current.count).toBe(10);
  });

  test("should increment count", () => {
    const { result } = renderHook(() => useCounter());

    act(() => {
      result.current.increment();
    });

    expect(result.current.count).toBe(1);
  });

  test("should decrement count", () => {
    const { result } = renderHook(() => useCounter(5));

    act(() => {
      result.current.decrement();
    });

    expect(result.current.count).toBe(4);
  });

  test("should reset count", () => {
    const { result } = renderHook(() => useCounter(10));

    act(() => {
      result.current.increment();
      result.current.reset();
    });

    expect(result.current.count).toBe(10);
  });
});
```

Testing with Context

```
// Test with context provider
function renderWithTheme(ui, { theme = 'light', ...renderOptions } = {}) {
  function Wrapper({ children }) {
    return (
```

```
      <ThemeProvider value={{ theme, toggleTheme: jest.fn() }}>
        {children}
      </ThemeProvider>
    );
  }

  return render(ui, { wrapper: Wrapper, ...renderOptions });
}

test('renders with theme context', () => {
  renderWithTheme(<ThemedComponent />);
  expect(screen.getByTestId('theme')).toHaveTextContent('light');
});

### Mocking Functions and Modules
```jsx
// Mocking fetch or API calls
import { render, screen, waitFor } from '@testing-library/react';
import App from './App';

global.fetch = jest.fn(() =>
  Promise.resolve({
    ok: true,
    json: () => Promise.resolve({ data: 'Hello' })
  })
);

test('fetches and displays data', async () => {
  render(<App />);
  await waitFor(() => expect(screen.getByText('Hello')).toBeInTheDocument());
});
```

Testing Async Logic

```
import { render, screen, fireEvent, waitFor } from "@testing-library/react";
import AsyncComponent from "./AsyncComponent";

test("shows loading and then data", async () => {
  render(<AsyncComponent />);
  expect(screen.getByText(/loading/i)).toBeInTheDocument();
  await waitFor(() =>
    expect(screen.getByText(/data loaded/i)).toBeInTheDocument()
  );
});
```

Code Coverage

```
# With Create React App
npm test -- --coverage
```
```

```
# With Vite + Vitest
npx vitest run --coverage
```

Useful Testing Tips

- Use `data-testid` for selecting elements when necessary, but prefer queries like `getByRole`, `getByLabelText`, or `getByText` for more robust tests.
- Mock timers with `jest.useFakeTimers()` for time-based logic.
- Use `act()` for updates that trigger React state changes outside of user events.
- Clean up after tests with `afterEach(cleanup)` if not handled automatically.
- Test accessibility: use `axe` or `jest-axe` to check for a11y issues.

Best Practices

1. Code Organization & Structure

- **Keep components small and focused.** Each should do one thing well.
- **Group by feature/folder, not by type.**

```
src/
  features/
    user/
      UserProfile.jsx
      userAPI.js
      userSlice.js
    product/
      ProductList.jsx
      productAPI.js
  components/
    Button.jsx
    Modal.jsx
  utils/
    formatDate.js
```

- **Use index.js for re-exports** to simplify imports.

2. Naming Conventions

- Use `PascalCase` for components, `camelCase` for functions/variables.
- File names should match the component name: `UserProfile.jsx`.
- Use clear, descriptive names for props and state.

3. State Management

- Use local state for UI, `useContext` or state libraries (Redux, Zustand) for global state.
- Avoid prop drilling by using context or composition.

- Keep state as flat as possible; avoid deeply nested objects.

4. Side Effects

- Use `useEffect` for side effects (fetching, subscriptions, timers).
- Always clean up subscriptions or timers in the cleanup function.
- Avoid unnecessary effects by specifying correct dependencies.

5. Performance Optimization

- Use `React.memo`, `useMemo`, and `useCallback` to prevent unnecessary re-renders.
- Lazy load heavy components with `React.lazy` and `Suspense`.
- Split code with dynamic imports for faster initial loads.
- Avoid inline functions/objects in props when possible.

6. Accessibility (a11y)

- Use semantic HTML elements (`<button>`, `<nav>`, `<main>`, etc.).
- Always provide `alt` text for images.
- Use labels for form fields and ensure keyboard navigation works.
- Test with screen readers and a11y tools.

7. Security

- Never trust user input; always validate and sanitize.
- Avoid dangerouslySetInnerHTML unless absolutely necessary.
- Store sensitive data securely (never in client-side code).
- Use HTTPS and secure cookies for authentication.

8. Testing

- Write tests for components, hooks, and utilities.
- Use React Testing Library for user-centric tests.
- Mock APIs and external dependencies.
- Aim for high code coverage, but focus on meaningful tests.

9. Code Quality

- Use a linter (ESLint) and formatter (Prettier) for consistent style.
- Use TypeScript or PropTypes for type safety.
- Review code with pull requests and code reviews.
- Document components and utilities with comments or Storybook.

10. Deployment & CI/CD

- Use environment variables for config (never hardcode secrets).
- Automate tests and builds with CI/CD pipelines (GitHub Actions, GitLab CI, etc.).
- Monitor performance and errors in production (Sentry, LogRocket, etc.).

This cheat sheet covers the essentials and advanced topics for React.js development. For more, check the React docs and keep practicing!