Complete Frontend Development Cheatsheet

From Beginner to Advanced - Your Ultimate Reference Guide

Table of Contents

Chapter 1: HTML Fundamentals

Basic Structure

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <!-- Content here -->
  </body>
</html>
```

Essential Elements

```
<!-- Headings -->
<h1>Main Title</h1>
<h2>Section Title</h2>
```

```
<h3>Subsection</h3>

<!-- Text Elements -->
<p>Paragraph text</p>
<span>Inline text</span>
<div>Block container</div>

<!-- Links and Navigation -->
<a href="https://example.com">External link</a>
<a href="#section">Internal link</a>
<nav>
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/about">About</a></li>
  </ul>
</nav>

<!-- Images and Media -->
<img src="image.jpg" alt="Description" loading="lazy" />
<video controls>
  <source src="video.mp4" type="video/mp4" />
</video>
<audio controls>
  <source src="audio.mp3" type="audio/mpeg" />
</audio>

<!-- Forms -->
<form action="/submit" method="POST">
  <input type="text" name="username" placeholder="Username" required />
  <input type="email" name="email" placeholder="Email" required />
  <input type="password" name="password" required />
  <select name="country">
    <option value="us">United States</option>
    <option value="ca">Canada</option>
  </select>
  <textarea name="message" rows="4"></textarea>
  <button type="submit">Submit</button>
</form>

<!-- Lists -->
<ul>
  <li>Unordered list item</li>
</ul>
<ol>
  <li>Ordered list item</li>
</ol>

<!-- Tables -->
<table>
  <thead>
    <tr>
      <th>Header 1</th>
      <th>Header 2</th>
    </tr>
```

```
  </thead>
  <tbody>
    <tr>
      <td>Data 1</td>
      <td>Data 2</td>
    </tr>
  </tbody>
</table>
```

Semantic HTML5

```
<header>
  <nav>Navigation</nav>
</header>
<main>
  <article>
    <section>
      <h2>Article Section</h2>
      <p>Content</p>
    </section>
  </article>
  <aside>Sidebar content</aside>
</main>
<footer>Footer content</footer>

<!-- Other semantic elements -->
<figure>
  <img src="chart.png" alt="Sales Chart" />
  <figcaption>Q1 Sales Data</figcaption>
</figure>

<details>
  <summary>Click to expand</summary>
  <p>Hidden content</p>
</details>

<time datetime="2024-01-01">January 1, 2024</time>
<mark>Highlighted text</mark>
<kbd>Ctrl+C</kbd>
<code>console.log()</code>
<pre><code>Preformatted code block</code></pre>
```

HTML Attributes

```
<!-- Global attributes -->
<div id="unique-id" class="class-name" data-custom="value">
  <p title="Tooltip text" lang="en" contenteditable="true">
    Editable paragraph
  </p>
```

```
</div>

<!-- Input attributes -->
<input
  type="text"
  name="username"
  id="username"
  placeholder="Enter username"
  required
  minlength="3"
  maxlength="20"
  pattern="[a-zA-Z0-9]+"
  autocomplete="username"
/>

<!-- Image attributes -->
<img
  src="image.jpg"
  alt="Alternative text"
  width="300"
  height="200"
  loading="lazy"
  srcset="image-small.jpg 300w, image-large.jpg 800w"
  sizes="(max-width: 600px) 300px, 800px"
/>
```

Chapter 2: CSS Mastery

CSS Basics

```
/* Selectors */
element {
} /* Type selector */
.class {
} /* Class selector */
#id {
} /* ID selector */
* {
} /* Universal selector */
element.class {
} /* Compound selector */
element,
.class {
} /* Group selector */

/* Descendant and child selectors */
.parent .child {
} /* Descendant */
.parent > .child {
} /* Direct child */
```

```
.element + .next {
} /* Adjacent sibling */
.element ~ .sibling {
} /* General sibling */

/* Pseudo-classes */
:hover,
:focus,
:active {
}
:first-child,
:last-child,
:nth-child(2n) {
}
:not(.class) {
}

/* Pseudo-elements */
::before,
::after {
}
::first-line,
::first-letter {
}
::placeholder,
::selection {
}
```

Box Model

```
.box {
  /* Box model properties */
  width: 300px;
  height: 200px;
  padding: 20px;
  border: 2px solid #000;
  margin: 10px;

  /* Box-sizing */
  box-sizing: border-box; /* Includes padding and border in width/height */
}

/* Display types */
.block {
  display: block;
}
.inline {
  display: inline;
}
.inline-block {
  display: inline-block;
```

```
}
.none {
  display: none;
}
```

Flexbox

```
.flex-container {
  display: flex;

  /* Direction */
  flex-direction: row | row-reverse | column | column-reverse;

  /* Wrapping */
  flex-wrap: nowrap | wrap | wrap-reverse;

  /* Shorthand */
  flex-flow: row wrap;

  /* Alignment */
  justify-content: flex-start | flex-end | center | space-between | space-around
    | space-evenly;
  align-items: stretch | flex-start | flex-end | center | baseline;
  align-content: flex-start | flex-end | center | space-between | space-around |
    stretch;

  /* Gap */
  gap: 20px;
  row-gap: 10px;
  column-gap: 15px;
}

.flex-item {
  /* Flex properties */
  flex-grow: 1; /* How much to grow */
  flex-shrink: 1; /* How much to shrink */
  flex-basis: auto; /* Initial size */
  flex: 1 1 auto; /* Shorthand */

  /* Individual alignment */
  align-self: auto | flex-start | flex-end | center | baseline | stretch;

  /* Order */
  order: 2;
}
```

CSS Grid

```css
.grid-container {
  display: grid;

  /* Define grid structure */
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: auto 1fr auto;

  /* Alternative syntax */
  grid-template-columns: repeat(3, 1fr);
  grid-template-columns: minmax(200px, 1fr) 2fr;

  /* Grid areas */
  grid-template-areas:
    "header header header"
    "sidebar main main"
    "footer footer footer";

  /* Gaps */
  gap: 20px;
  row-gap: 10px;
  column-gap: 15px;

  /* Alignment */
  justify-items: start | end | center | stretch;
  align-items: start | end | center | stretch;
  justify-content: start | end | center | stretch | space-around | space-between
    | space-evenly;
  align-content: start | end | center | stretch | space-around | space-between |
    space-evenly;
}

.grid-item {
  /* Grid positioning */
  grid-column: 1 / 3; /* Start at line 1, end at line 3 */
  grid-row: 2 / 4;

  /* Alternative syntax */
  grid-column-start: 1;
  grid-column-end: 3;
  grid-row-start: 2;
  grid-row-end: 4;

  /* Named areas */
  grid-area: header;

  /* Individual alignment */
  justify-self: start | end | center | stretch;
  align-self: start | end | center | stretch;
}
```

Positioning

```
.positioned {
  position: static | relative | absolute | fixed | sticky;

  /* Offset properties */
  top: 10px;
  right: 20px;
  bottom: 30px;
  left: 40px;

  /* Z-index for stacking */
  z-index: 1000;
}

/* Common positioning patterns */
.relative-parent {
  position: relative;
}

.absolute-child {
  position: absolute;
  top: 0;
  left: 0;
}

.fixed-header {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  z-index: 1000;
}

.sticky-nav {
  position: sticky;
  top: 0;
}
```

Responsive Design

```
/* Media queries */
@media screen and (max-width: 768px) {
  .container {
    width: 100%;
    padding: 10px;
  }
}

@media screen and (min-width: 769px) and (max-width: 1024px) {
  .container {
    width: 90%;
```

```
  }
}

/* Common breakpoints */
@media (max-width: 480px) {
  /* Mobile */
}
@media (min-width: 481px) and (max-width: 768px) {
  /* Tablet */
}
@media (min-width: 769px) and (max-width: 1024px) {
  /* Desktop */
}
@media (min-width: 1025px) {
  /* Large Desktop */
}

/* Responsive units and functions */
.responsive {
  width: 100%;
  max-width: 1200px;
  font-size: clamp(1rem, 2.5vw, 2rem); /* Responsive font size */
  padding: min(5%, 2rem); /* Responsive padding */
}

/* CSS Variables for theming */
:root {
  --primary-color: #007bff;
  --secondary-color: #6c757d;
  --font-size-base: 1rem;
  --spacing-unit: 8px;
}

.themed-element {
  color: var(--primary-color);
  font-size: var(--font-size-base);
  margin: calc(var(--spacing-unit) * 2);
}
```

Advanced CSS

```
/* Transforms */
.transformed {
  transform: translateX(100px) translateY(50px);
  transform: rotate(45deg);
  transform: scale(1.5);
  transform: skew(15deg, 10deg);
  transform: matrix(1, 0.5, -0.5, 1, 100, 50);

  /* 3D transforms */
  transform: perspective(1000px) rotateX(45deg) rotateY(45deg);
```

```
  transform-style: preserve-3d;
  backface-visibility: hidden;
}

/* Transitions */
.smooth-transition {
  transition: all 0.3s ease-in-out;
  transition: opacity 0.3s ease, transform 0.5s cubic-bezier(0.25, 0.46, 0.45,
0.94);
}

/* Animations */
@keyframes slideIn {
  from {
    opacity: 0;
    transform: translateX(-100%);
  }
  to {
    opacity: 1;
    transform: translateX(0);
  }
}

.animated {
  animation: slideIn 0.5s ease-out;
  animation: slideIn 0.5s ease-out 0.2s both infinite alternate;
  /* name | duration | timing-function | delay | fill-mode | iteration-count |
direction */
}

/* Filters and effects */
.filtered {
  filter: blur(5px) brightness(0.8) contrast(1.2) hue-rotate(90deg) saturate(1.5);
  backdrop-filter: blur(10px) brightness(0.8);
}

/* Modern CSS features */
.modern {
  /* Container queries */
  container-type: inline-size;

  /* CSS shapes */
  shape-outside: circle(50%);
  clip-path: polygon(50% 0%, 0% 100%, 100% 100%);

  /* CSS logical properties */
  margin-inline: auto;
  padding-block: 1rem;
  border-inline-start: 2px solid blue;

  /* CSS grid subgrid */
  display: subgrid;
  grid-template-columns: subgrid;
}
```

```
@container (min-width: 400px) {
  .card {
    display: grid;
    grid-template-columns: 1fr 2fr;
  }
}
```

Chapter 3: JavaScript Deep Dive

JavaScript Fundamentals

```
// Variables and data types
const constant = "immutable";
let variable = "mutable";
var oldStyle = "function-scoped";

// Primitive types
const string = "Hello World";
const number = 42;
const boolean = true;
const nullValue = null;
const undefined = undefined;
const symbol = Symbol("unique");
const bigint = 123n;

// Objects and arrays
const object = {
  key: "value",
  method() {
    return this.key;
  },
};

const array = [1, 2, 3, 4, 5];

// Functions
function declaration() {
  return "function declaration";
}

const expression = function () {
  return "function expression";
};

const arrow = () => "arrow function";

const arrowComplex = (param1, param2) => {
  // Complex logic
```

```
  return param1 + param2;
};
```

Modern ES6+ Features

```
// Destructuring
const { name, age, ...rest } = person;
const [first, second, ...remaining] = array;

// Template literals
const message = `Hello ${name}, you are ${age} years old`;

// Spread operator
const newArray = [...array1, ...array2];
const newObject = { ...object1, ...object2 };

// Default parameters
function greet(name = "World") {
  return `Hello ${name}`;
}

// Rest parameters
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}

// Enhanced object literals
const obj = {
  name,
  age,
  [computedKey]: "computed value",
  method() {
    return "method shorthand";
  },
};

// Classes
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    return `Hello, I'm ${this.name}`;
  }

  static species() {
    return "Homo sapiens";
  }
}
```

```
class Employee extends Person {
  constructor(name, age, position) {
    super(name, age);
    this.position = position;
  }

  work() {
    return `${this.name} is working as ${this.position}`;
  }
}

// Modules
export const constant = "exported constant";
export function exportedFunction() {}
export default class DefaultExport {}

import DefaultExport, { constant, exportedFunction } from "./module.js";
import * as Module from "./module.js";
```

Asynchronous JavaScript

```
// Promises
const promise = new Promise((resolve, reject) => {
  if (success) {
    resolve("Success!");
  } else {
    reject(new Error("Failure!"));
  }
});

promise
  .then((result) => console.log(result))
  .catch((error) => console.error(error))
  .finally(() => console.log("Cleanup"));

// Async/Await
async function fetchData() {
  try {
    const response = await fetch("/api/data");
    const data = await response.json();
    return data;
  } catch (error) {
    console.error("Error fetching data:", error);
    throw error;
  }
}

// Promise utilities
Promise.all([promise1, promise2, promise3]).then((results) =>
  console.log("All promises resolved")
```

```
);

Promise.allSettled([promise1, promise2, promise3]).then((results) =>
  console.log("All promises settled")
);

Promise.race([promise1, promise2, promise3]).then((result) =>
  console.log("First promise resolved")
);

// Async iteration
async function* asyncGenerator() {
  yield await fetchData();
  yield await fetchMoreData();
}

for await (const data of asyncGenerator()) {
  console.log(data);
}
```

DOM Manipulation

```
// Element selection
const element = document.getElementById("id");
const elements = document.getElementsByClassName("class");
const element = document.querySelector(".class");
const elements = document.querySelectorAll(".class");

// Element creation and manipulation
const newElement = document.createElement("div");
newElement.textContent = "Hello World";
newElement.innerHTML = "<strong>Bold text</strong>";
newElement.className = "my-class";
newElement.setAttribute("data-id", "123");

// DOM traversal
const parent = element.parentNode;
const children = element.children;
const firstChild = element.firstElementChild;
const lastChild = element.lastElementChild;
const nextSibling = element.nextElementSibling;
const previousSibling = element.previousElementSibling;

// Event handling
element.addEventListener("click", function (event) {
  event.preventDefault();
  event.stopPropagation();
  console.log("Element clicked");
});

// Modern event handling patterns
```

```
element.addEventListener("click", (event) => {
  const { target, currentTarget, type } = event;
  console.log(`${type} event on`, target);
});

// Event delegation
document.addEventListener("click", (event) => {
  if (event.target.matches(".button")) {
    handleButtonClick(event);
  }
});

// Custom events
const customEvent = new CustomEvent("myEvent", {
  detail: { message: "Custom event data" },
  bubbles: true,
});

element.dispatchEvent(customEvent);

element.addEventListener("myEvent", (event) => {
  console.log(event.detail.message);
});
```

Advanced JavaScript Concepts

```
// Closures
function createCounter() {
  let count = 0;
  return function () {
    count++;
    return count;
  };
}

const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2

// Higher-order functions
const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map((n) => n * 2);
const evens = numbers.filter((n) => n % 2 === 0);
const sum = numbers.reduce((total, n) => total + n, 0);

// Function composition
const compose = (f, g) => (x) => f(g(x));
const pipe = (...fns) => (x) => fns.reduce((v, f) => f(v), x);

// Currying
```

```
const curry = (fn) => {
  return function curried(...args) {
    if (args.length >= fn.length) {
      return fn.apply(this, args);
    } else {
      return function (...args2) {
        return curried.apply(this, args.concat(args2));
      };
    }
  };
};

// Debouncing and throttling
function debounce(func, wait) {
  let timeout;
  return function executedFunction(...args) {
    const later = () => {
      clearTimeout(timeout);
      func(...args);
    };
    clearTimeout(timeout);
    timeout = setTimeout(later, wait);
  };
}

function throttle(func, limit) {
  let inThrottle;
  return function () {
    const args = arguments;
    const context = this;
    if (!inThrottle) {
      func.apply(context, args);
      inThrottle = true;
      setTimeout(() => (inThrottle = false), limit);
    }
  };
}

// Memoization
function memoize(fn) {
  const cache = new Map();
  return function (...args) {
    const key = JSON.stringify(args);
    if (cache.has(key)) {
      return cache.get(key);
    }
    const result = fn.apply(this, args);
    cache.set(key, result);
    return result;
  };
}

// Observer pattern
class EventEmitter {
```

```
  constructor() {
    this.events = {};
  }

  on(event, callback) {
    if (!this.events[event]) {
      this.events[event] = [];
    }
    this.events[event].push(callback);
  }

  emit(event, data) {
    if (this.events[event]) {
      this.events[event].forEach((callback) => callback(data));
    }
  }

  off(event, callback) {
    if (this.events[event]) {
      this.events[event] = this.events[event].filter((cb) => cb !== callback);
    }
  }
}
```

Error Handling

```
// Try-catch blocks
try {
  const result = riskyOperation();
  console.log(result);
} catch (error) {
  if (error instanceof TypeError) {
    console.error("Type error:", error.message);
  } else if (error instanceof ReferenceError) {
    console.error("Reference error:", error.message);
  } else {
    console.error("Unknown error:", error);
  }
} finally {
  console.log("Cleanup code");
}

// Custom errors
class CustomError extends Error {
  constructor(message, code) {
    super(message);
    this.name = "CustomError";
    this.code = code;
  }
}
```

```
// Error handling with async/await
async function handleAsyncErrors() {
  try {
    const data = await fetchData();
    return data;
  } catch (error) {
    console.error("Async error:", error);
    throw new CustomError("Failed to fetch data", "FETCH_ERROR");
  }
}

// Global error handling
window.addEventListener("error", (event) => {
  console.error("Global error:", event.error);
});

window.addEventListener("unhandledrejection", (event) => {
  console.error("Unhandled promise rejection:", event.reason);
  event.preventDefault();
});
```

Chapter 4: React.js Complete Guide

React Basics

```
// Functional component
import React from "react";

const MyComponent = () => {
  return (
    <div>
      <h1>Hello, React!</h1>
    </div>
  );
};

export default MyComponent;

// Component with props
const Greeting = ({ name, age = 0 }) => {
  return (
    <div>
      <h1>Hello, {name}!</h1>
      {age > 0 && <p>You are {age} years old.</p>}
    </div>
  );
};

// Usage
<Greeting name="John" age={25} />;
```

React Hooks

```
import React, {
  useState,
  useEffect,
  useContext,
  useReducer,
  useCallback,
  useMemo,
  useRef,
} from "react";

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

  const increment = () => setCount(count + 1);
  const decrement = () => setCount((prev) => prev - 1); // Functional update

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
    </div>
  );
};

// useEffect
const DataFetcher = ({ userId }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

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

    const fetchUser = async () => {
      try {
        setLoading(true);
        const response = await fetch(`/api/users/${userId}`);
        const userData = await response.json();

        if (!cancelled) {
          setUser(userData);
        }
      } catch (err) {
        if (!cancelled) {
          setError(err.message);
        }
      } finally {
```

```
        if (!cancelled) {
          setLoading(false);
        }
      }
    };

    fetchUser();

    return () => {
      cancelled = true; // Cleanup
    };
  }, [userId]); // Dependency array

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;
  if (!user) return <div>User not found</div>;

  return <div>Welcome, {user.name}!</div>;
};

// useContext
const ThemeContext = React.createContext();

const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState("light");

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

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

const ThemedComponent = () => {
  const { theme, toggleTheme } = useContext(ThemeContext);

  return (
    <div className={`theme-${theme}`}>
      <p>Current theme: {theme}</p>
      <button onClick={toggleTheme}>Toggle Theme</button>
    </div>
  );
};

// useReducer
const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case "increment":
```

```
      return { count: state.count + 1 };
    case "decrement":
      return { count: state.count - 1 };
    case "reset":
      return initialState;
    default:
      throw new Error();
  }
}

const ComplexCounter = () => {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      Count: {state.count}
      <button onClick={() => dispatch({ type: "increment" })}>+</button>
      <button onClick={() => dispatch({ type: "decrement" })}>-</button>
      <button onClick={() => dispatch({ type: "reset" })}>Reset</button>
    </div>
  );
};

// useCallback and useMemo
const ExpensiveComponent = ({ items, filter }) => {
  const expensiveValue = useMemo(() => {
    return items
      .filter((item) => item.includes(filter))
      .reduce((acc, item) => acc + item.length, 0);
  }, [items, filter]);

  const handleClick = useCallback((id) => {
    console.log("Clicked item:", id);
  }, []);

  return (
    <div>
      <p>Expensive value: {expensiveValue}</p>
      {items.map((item) => (
        <button key={item} onClick={() => handleClick(item)}>
          {item}
        </button>
      ))}
    </div>
  );
};

// useRef
const FocusInput = () => {
  const inputRef = useRef(null);

  const focusInput = () => {
    inputRef.current.focus();
  };
```

```
  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
};
```

Custom Hooks

```
// Custom hook for API calls
const useApi = (url) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true);
        const response = await fetch(url);
        const result = await response.json();
        setData(result);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error };
};

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

  const setValue = (value) => {
    try {
      setStoredValue(value);
```

```
      window.localStorage.setItem(key, JSON.stringify(value));
    } catch (error) {
      console.error(error);
    }
  };

  return [storedValue, setValue];
};

// Custom hook for window size
const useWindowSize = () => {
  const [windowSize, setWindowSize] = useState({
    width: undefined,
    height: undefined,
  });

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

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

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

  return windowSize;
};

// Usage of custom hooks
const MyComponent = () => {
  const { data, loading, error } = useApi("/api/data");
  const [theme, setTheme] = useLocalStorage("theme", "light");
  const { width, height } = useWindowSize();

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

  return (
    <div>
      <p>
        Window size: {width} x {height}
      </p>
      <p>Theme: {theme}</p>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
};
```

React Patterns

```
// Higher-Order Components (HOC)
const withLoading = (Component) => {
  return function WithLoadingComponent({ isLoading, ...props }) {
    if (isLoading) {
      return <div>Loading...</div>;
    }
    return <Component {...props} />;
  };
};

const EnhancedComponent = withLoading(MyComponent);

// Render Props Pattern
const MouseTracker = ({ render }) => {
  const [position, setPosition] = useState({ x: 0, y: 0 });

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

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

  return render(position);
};

// Usage
<MouseTracker
  render={({ x, y }) => (
    <h1>
      Mouse position: {x}, {y}
    </h1>
  )}
/>;

// Compound Components Pattern
const Tabs = ({ children, defaultTab = 0 }) => {
  const [activeTab, setActiveTab] = useState(defaultTab);

  return (
    <div className="tabs">
      {React.Children.map(children, (child, index) =>
        React.cloneElement(child, { activeTab, setActiveTab, index })
      )}
    </div>
  );
};
```

```
const TabList = ({ children, activeTab, setActiveTab }) => (
  <div className="tab-list">
    {React.Children.map(children, (child, index) =>
      React.cloneElement(child, {
        isActive: activeTab === index,
        onClick: () => setActiveTab(index),
      })
    )}
  </div>
);

const Tab = ({ children, isActive, onClick }) => (
  <button className={`tab ${isActive ? "active" : ""}`} onClick={onClick}>
    {children}
  </button>
);

const TabPanels = ({ children, activeTab }) => (
  <div className="tab-panels">
    {React.Children.toArray(children)[activeTab]}
  </div>
);

// Usage
<Tabs defaultTab={0}>
  <TabList>
    <Tab>Tab 1</Tab>
    <Tab>Tab 2</Tab>
    <Tab>Tab 3</Tab>
  </TabList>
  <TabPanels>
    <div>Panel 1 Content</div>
    <div>Panel 2 Content</div>
    <div>Panel 3 Content</div>
  </TabPanels>
</Tabs>;
```

React Performance Optimization

```
import React, { memo, useMemo, useCallback, lazy, Suspense } from "react";

// React.memo for component memoization
const ExpensiveComponent = memo(({ data, onUpdate }) => {
  console.log("ExpensiveComponent rendered");

  return (
    <div>
      {data.map((item) => (
        <div key={item.id}>{item.name}</div>
      ))}
    </div>
```

```
  );
});

// Custom comparison function
const MyComponent = memo(
  ({ user, posts }) => {
    return <div>...</div>;
  },
  (prevProps, nextProps) => {
    return (
      prevProps.user.id === nextProps.user.id &&
      prevProps.posts.length === nextProps.posts.length
    );
  }
);

// Code splitting with lazy loading
const LazyComponent = lazy(() => import("./LazyComponent"));

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

// Virtualization for large lists
import { FixedSizeList as List } from "react-window";

const VirtualizedList = ({ items }) => {
  const Row = ({ index, style }) => (
    <div style={style}>{items[index].name}</div>
  );

  return (
    <List height={600} itemCount={items.length} itemSize={35} width="100%">
      {Row}
    </List>
  );
};
```

React Router

```
import {
  BrowserRouter,
  Routes,
  Route,
  Link,
  useNavigate,
  useParams,
```

```
  useLocation,
} from "react-router-dom";

// Basic routing setup
const App = () => {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
        <Link to="/users">Users</Link>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/users" element={<Users />} />
        <Route path="/users/:id" element={<UserDetail />} />
        <Route
          path="/protected"
          element={
            <ProtectedRoute>
              <Dashboard />
            </ProtectedRoute>
          }
        />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
};

// Route parameters
const UserDetail = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const location = useLocation();

  const goBack = () => {
    navigate(-1);
  };

  const goToUsers = () => {
    navigate("/users", { replace: true });
  };

  return (
    <div>
      <h1>User {id}</h1>
      <p>Current path: {location.pathname}</p>
      <button onClick={goBack}>Go Back</button>
      <button onClick={goToUsers}>Go to Users</button>
    </div>
  );
```

```
};

// Protected routes
const ProtectedRoute = ({ children }) => {
  const isAuthenticated = useAuth();
  const location = useLocation();

  if (!isAuthenticated) {
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  return children;
};

// Nested routes
const Users = () => {
  return (
    <div>
      <h1>Users</h1>
      <Routes>
        <Route index element={<UsersList />} />
        <Route path=":id" element={<UserDetail />} />
        <Route path="new" element={<NewUser />} />
      </Routes>
    </div>
  );
};
```

Form Handling

```
// Controlled components
const ContactForm = () => {
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    message: "",
  });
  const [errors, setErrors] = useState({});

  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]: "",
```

```
      }));
    }
  };

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

    if (!formData.name.trim()) {
      newErrors.name = "Name is required";
    }

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

    if (!formData.message.trim()) {
      newErrors.message = "Message is required";
    }

    return newErrors;
  };

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

    const newErrors = validateForm();
    if (Object.keys(newErrors).length > 0) {
      setErrors(newErrors);
      return;
    }

    // Submit form
    console.log("Form submitted:", formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor="name">Name:</label>
        <input
          type="text"
          id="name"
          name="name"
          value={formData.name}
          onChange={handleChange}
          required
        />
        {errors.name && <span className="error">{errors.name}</span>}
      </div>

      <div>
        <label htmlFor="email">Email:</label>
```

```
          <input
            type="email"
            id="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
            required
          />
          {errors.email && <span className="error">{errors.email}</span>}
        </div>

        <div>
          <label htmlFor="message">Message:</label>
          <textarea
            id="message"
            name="message"
            value={formData.message}
            onChange={handleChange}
            required
          />
          {errors.message && <span className="error">{errors.message}</span>}
        </div>

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

// Using react-hook-form
import { useForm } from "react-hook-form";

const HookForm = () => {
  const {
    register,
    handleSubmit,
    formState: { errors },
    watch,
    setValue,
    reset,
  } = useForm({
    defaultValues: {
      name: "",
      email: "",
      age: 0,
    },
  });

  const watchedName = watch("name");

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

  return (
```

```
      <form onSubmit={handleSubmit(onSubmit)}>
        <input
          {...register("name", {
            required: "Name is required",
            minLength: {
              value: 2,
              message: "Name must be at least 2 characters",
            },
          })}
          placeholder="Name"
        />
        {errors.name && <span>{errors.name.message}</span>}

        <input
          {...register("email", {
            required: "Email is required",
            pattern: {
              value: /\S+@\S+\.\S+/,
              message: "Invalid email address",
            },
          })}
          placeholder="Email"
          type="email"
        />
        {errors.email && <span>{errors.email.message}</span>}

        <input
          {...register("age", {
            required: "Age is required",
            min: {
              value: 1,
              message: "Age must be positive",
            },
            max: {
              value: 120,
              message: "Age must be realistic",
            },
          })}
          placeholder="Age"
          type="number"
        />
        {errors.age && <span>{errors.age.message}</span>}

        <button type="submit">Submit</button>
        <button type="button" onClick={() => reset()}>
          Reset
        </button>
      </form>
    );
  };
```

Chapter 5: Next.js Framework

Next.js Basics

```
// pages/index.js - Home page
import Head from "next/head";
import Link from "next/link";
import Image from "next/image";

export default function Home({ posts }) {
  return (
    <div>
      <Head>
        <title>My Next.js App</title>
        <meta name="description" content="Generated by Next.js" />
        <link rel="icon" href="/favicon.ico" />
      </Head>

      <main>
        <h1>Welcome to Next.js!</h1>

        <nav>
          <Link href="/about">
            <a>About</a>
          </Link>
          <Link href="/blog">
            <a>Blog</a>
          </Link>
        </nav>

        <div>
          {posts.map((post) => (
            <article key={post.id}>
              <h2>
                <Link href={`/blog/${post.slug}`}>
                  <a>{post.title}</a>
                </Link>
              </h2>
              <p>{post.excerpt}</p>
            </article>
          ))}
        </div>

        <Image
          src="/hero-image.jpg"
          alt="Hero Image"
          width={800}
          height={400}
          priority
        />
      </main>
    </div>
```

```
  );
}

// Static generation
export async function getStaticProps() {
  const posts = await fetchPosts();

  return {
    props: {
      posts,
    },
    revalidate: 60, // ISR - regenerate every 60 seconds
  };
}
```

Next.js Routing

```
// pages/blog/[slug].js - Dynamic routing
import { useRouter } from "next/router";
import ErrorPage from "next/error";

export default function BlogPost({ post }) {
  const router = useRouter();

  if (!router.isFallback && !post?.slug) {
    return <ErrorPage statusCode={404} />;
  }

  if (router.isFallback) {
    return <div>Loading...</div>;
  }

  return (
    <article>
      <h1>{post.title}</h1>
      <div dangerouslySetInnerHTML={{ __html: post.content }} />
    </article>
  );
}

// pages/blog/[...params].js - Catch-all routes
export default function CatchAll() {
  const router = useRouter();
  const { params } = router.query;

  return (
    <div>
      <h1>Catch All Route</h1>
      <p>Params: {JSON.stringify(params)}</p>
    </div>
  );
```

```
}

// pages/api/posts/[id].js - API routes
export default function handler(req, res) {
  const { id } = req.query;
  const { method } = req;

  switch (method) {
    case "GET":
      const post = getPostById(id);
      res.status(200).json(post);
      break;

    case "PUT":
      const updatedPost = updatePost(id, req.body);
      res.status(200).json(updatedPost);
      break;

    case "DELETE":
      deletePost(id);
      res.status(204).end();
      break;

    default:
      res.setHeader("Allow", ["GET", "PUT", "DELETE"]);
      res.status(405).end(`Method ${method} Not Allowed`);
  }
}
```

Data Fetching

```
// Static Generation with getStaticProps
export async function getStaticProps(context) {
  const { params, preview, previewData } = context;

  try {
    const data = await fetchData(params.id);

    return {
      props: {
        data,
      },
      revalidate: 3600, // ISR
    };
  } catch (error) {
    return {
      notFound: true,
    };
  }
}
```

```
// Dynamic paths with getStaticPaths
export async function getStaticPaths() {
  const posts = await fetchAllPosts();

  const paths = posts.map((post) => ({
    params: { slug: post.slug },
  }));

  return {
    paths,
    fallback: "blocking", // true, false, or 'blocking'
  };
}

// Server-side rendering with getServerSideProps
export async function getServerSideProps(context) {
  const { req, res, params, query } = context;

  // Access cookies
  const token = req.cookies.token;

  if (!token) {
    return {
      redirect: {
        destination: "/login",
        permanent: false,
      },
    };
  }

  const user = await fetchUser(token);

  return {
    props: {
      user,
    },
  };
}

// Client-side data fetching with SWR
import useSWR from "swr";

const fetcher = (url) => fetch(url).then((res) => res.json());

function Profile() {
  const { data, error, mutate } = useSWR("/api/user", fetcher);

  if (error) return <div>Failed to load</div>;
  if (!data) return <div>Loading...</div>;

  return (
    <div>
      <h1>Hello {data.name}!</h1>
      <button onClick={() => mutate()}>Refresh</button>
```

```
    </div>
  );
}
```

Next.js App Router (13+)

```
// app/layout.js - Root layout
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <header>
          <nav>Navigation</nav>
        </header>
        <main>{children}</main>
        <footer>Footer</footer>
      </body>
    </html>
  );
}

// app/page.js - Home page
export default function HomePage() {
  return <h1>Home Page</h1>;
}

// app/blog/[slug]/page.js - Dynamic route
export default function BlogPost({ params }) {
  return <h1>Post: {params.slug}</h1>;
}

// app/blog/loading.js - Loading UI
export default function Loading() {
  return <div>Loading blog posts...</div>;
}

// app/blog/error.js - Error UI
("use client");

export default function Error({ error, reset }) {
  return (
    <div>
      <h2>Something went wrong!</h2>
      <button onClick={() => reset()}>Try again</button>
    </div>
  );
}

// app/api/posts/route.js - API route
export async function GET(request) {
  const posts = await fetchPosts();
```

```
  return Response.json(posts);
}

export async function POST(request) {
  const body = await request.json();
  const post = await createPost(body);
  return Response.json(post, { status: 201 });
}

// Server Components vs Client Components
// Server Component (default)
async function ServerComponent() {
  const data = await fetchData(); // Can use async/await

  return (
    <div>
      <h1>Server Component</h1>
      <p>{data.message}</p>
    </div>
  );
}

// Client Component
("use client");

function ClientComponent() {
  const [count, setCount] = useState(0); // Can use hooks

  return (
    <div>
      <h1>Client Component</h1>
      <button onClick={() => setCount(count + 1)}>Count: {count}</button>
    </div>
  );
}
```

Next.js Configuration

```
// next.config.js
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  swcMinify: true,

  // Image optimization
  images: {
    domains: ["example.com", "images.unsplash.com"],
    formats: ["image/webp", "image/avif"],
  },

  // Environment variables
```

```
  env: {
    CUSTOM_KEY: "my-value",
  },

  // Redirects
  async redirects() {
    return [
      {
        source: "/old-page",
        destination: "/new-page",
        permanent: true,
      },
    ];
  },

  // Rewrites
  async rewrites() {
    return [
      {
        source: "/api/:path*",
        destination: "https://api.example.com/:path*",
      },
    ];
  },

  // Headers
  async headers() {
    return [
      {
        source: "/(.*)",
        headers: [
          {
            key: "X-Content-Type-Options",
            value: "nosniff",
          },
        ],
      },
    ];
  },

  // Webpack customization
  webpack: (config, { buildId, dev, isServer, defaultLoaders, webpack }) => {
    // Custom webpack config
    return config;
  },
};

module.exports = nextConfig;
```

Chapter 6: TypeScript Integration

TypeScript Basics

```typescript
// Basic types
let name: string = "John";
let age: number = 30;
let isActive: boolean = true;
let hobbies: string[] = ["reading", "gaming"];
let coordinates: [number, number] = [10, 20]; // Tuple

// Objects and interfaces
interface User {
  id: number;
  name: string;
  email: string;
  age?: number; // Optional property
  readonly createdAt: Date; // Readonly property
}

const user: User = {
  id: 1,
  name: "John Doe",
  email: "john@example.com",
  createdAt: new Date(),
};

// Union types
type Status = "pending" | "approved" | "rejected";
let currentStatus: Status = "pending";

// Functions
function greet(name: string, age?: number): string {
  return age ? `Hello ${name}, age ${age}` : `Hello ${name}`;
}

const calculateArea = (width: number, height: number): number => {
  return width * height;
};

// Function types
type MathOperation = (a: number, b: number) => number;

const add: MathOperation = (a, b) => a + b;
const subtract: MathOperation = (a, b) => a - b;

// Generic types
function identity<T>(arg: T): T {
  return arg;
}

interface ApiResponse<T> {
  data: T;
  status: number;
```

```
  message: string;
}

const userResponse: ApiResponse<User> = {
  data: user,
  status: 200,
  message: "Success",
};

// Classes
class Animal {
  protected name: string;

  constructor(name: string) {
    this.name = name;
  }

  public speak(): string {
    return `${this.name} makes a sound`;
  }
}

class Dog extends Animal {
  private breed: string;

  constructor(name: string, breed: string) {
    super(name);
    this.breed = breed;
  }

  public speak(): string {
    return `${this.name} barks`;
  }

  public getBreed(): string {
    return this.breed;
  }
}

// Enums
enum Color {
  Red = "#ff0000",
  Green = "#00ff00",
  Blue = "#0000ff",
}

enum HttpStatus {
  OK = 200,
  NotFound = 404,
  InternalServerError = 500,
}

// Advanced types
type Partial<T> = {
```

```
  [P in keyof T]?: T[P];
};

type Pick<T, K extends keyof T> = {
  [P in K]: T[P];
};

type UserUpdate = Partial<User>;
type UserSummary = Pick<User, "id" | "name">;

// Conditional types
type NonNullable<T> = T extends null | undefined ? never : T;

// Utility types
interface CreateUserRequest {
  name: string;
  email: string;
  password: string;
}

type UpdateUserRequest = Partial<CreateUserRequest>;
type UserResponse = Omit<CreateUserRequest, "password">;
type RequiredUser = Required<User>;
```

React with TypeScript

```
import React, { useState, useEffect, ReactNode, FC } from "react";

// Component props interface
interface ButtonProps {
  children: ReactNode;
  onClick: () => void;
  variant?: "primary" | "secondary";
  disabled?: boolean;
  className?: string;
}

// Functional component with TypeScript
const Button: FC<ButtonProps> = ({
  children,
  onClick,
  variant = "primary",
  disabled = false,
  className = "",
}) => {
  return (
    <button
      className={`btn btn-${variant} ${className}`}
      onClick={onClick}
      disabled={disabled}
    >
```

```
      {children}
    </button>
  );
};

// Component with hooks
interface User {
  id: number;
  name: string;
  email: string;
}

const UserList: FC = () => {
  const [users, setUsers] = useState<User[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        const response = await fetch("/api/users");
        const data: User[] = await response.json();
        setUsers(data);
      } catch (err) {
        setError(err instanceof Error ? err.message : "Unknown error");
      } finally {
        setLoading(false);
      }
    };

    fetchUsers();
  }, []);

  const handleUserClick = (user: User): void => {
    console.log("Clicked user:", user);
  };

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

  return (
    <div>
      {users.map((user) => (
        <div key={user.id} onClick={() => handleUserClick(user)}>
          {user.name} - {user.email}
        </div>
      ))}
    </div>
  );
};

// Generic component
interface ListProps<T> {
  items: T[];
```

```
  renderItem: (item: T) => ReactNode;
  keyExtractor: (item: T) => string | number;
}

function List<T>({ items, renderItem, keyExtractor }: ListProps<T>) {
  return (
    <div>
      {items.map((item) => (
        <div key={keyExtractor(item)}>{renderItem(item)}</div>
      ))}
    </div>
  );
}

// Usage
<List
  items={users}
  renderItem={(user) => <span>{user.name}</span>}
  keyExtractor={(user) => user.id}
/>;

// Custom hooks with TypeScript
interface UseApiResult<T> {
  data: T | null;
  loading: boolean;
  error: string | null;
  refetch: () => void;
}

function useApi<T>(url: string): UseApiResult<T> {
  const [data, setData] = useState<T | null>(null);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  const fetchData = async () => {
    try {
      setLoading(true);
      const response = await fetch(url);
      const result: T = await response.json();
      setData(result);
    } catch (err) {
      setError(err instanceof Error ? err.message : "Unknown error");
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchData();
  }, [url]);

  return { data, loading, error, refetch: fetchData };
}
```

```
// Context with TypeScript
interface AuthContextType {
  user: User | null;
  login: (email: string, password: string) => Promise<void>;
  logout: () => void;
  loading: boolean;
}

const AuthContext = React.createContext<AuthContextType | undefined>(undefined);

export const useAuth = (): AuthContextType => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error("useAuth must be used within an AuthProvider");
  }
  return context;
};

const AuthProvider: FC<{ children: ReactNode }> = ({ children }) => {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  const login = async (email: string, password: string): Promise<void> => {
    // Login logic
  };

  const logout = (): void => {
    setUser(null);
  };

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

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

Next.js with TypeScript

```
// types/index.ts
export interface Post {
    id: number;
    title: string;
    content: string;
    slug: string;
    publishedAt: string;
    author: {
        id: number;
```

```
        name: string;
    };
}

export interface ApiResponse<T> {
    data: T;
    status: number;
    message: string;
}

// pages/blog/[slug].tsx
import { GetStaticProps, GetStaticPaths, NextPage } from 'next';
import { Post } from '../../types';

interface BlogPostProps {
    post: Post;
}

const BlogPost: NextPage<BlogPostProps> = ({ post }) => {
    return (
        <article>
            <h1>{post.title}</h1>
            <div dangerouslySetInnerHTML={{ __html: post.content }} />
        </article>
    );
};

export const getStaticPaths: GetStaticPaths = async () => {
    const posts = await fetchAllPosts();

    const paths = posts.map((post: Post) => ({
        params: { slug: post.slug },
    }));

    return {
        paths,
        fallback: false,
    };
};

export const getStaticProps: GetStaticProps<BlogPostProps> = async ({ params }) =>
{
    const slug = params?.slug as string;
    const post = await fetchPostBySlug(slug);

    if (!post) {
        return {
            notFound: true,
        };
    }

    return {
        props: {
            post,
```

```
        },
        revalidate: 60,
    };
};

export default BlogPost;

// pages/api/posts/[id].ts
import { NextApiRequest, NextApiResponse } from 'next';
import { Post } from '../../../types';

interface PostRequest extends NextApiRequest {
```

Chapter 7: State Management

Introduction

State management is crucial for building scalable frontend applications. It helps manage and synchronize data across components, handle side effects, and maintain UI consistency.

Local State (React)

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

const Counter = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <button onClick={() => setCount(count - 1)}>-</button>
      <span>{count}</span>
      <button onClick={() => setCount(count + 1)}>+</button>
    </div>
  );
};
```

Lifting State Up

```
const Parent = () => {
  const [value, setValue] = useState("");
  return <Child value={value} onChange={setValue} />;
};

const Child = ({ value, onChange }) => (
  <input value={value} onChange={(e) => onChange(e.target.value)} />
);
```

Context API

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

const ThemeContext = createContext();

const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState("light");
  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

const useTheme = () => useContext(ThemeContext);

// Usage
const ThemedButton = () => {
  const { theme, setTheme } = useTheme();
  return (
    <button onClick={() => setTheme(theme === "light" ? "dark" : "light")}>
      Current theme: {theme}
    </button>
  );
};
```

Redux (Global State)

```
// store.js
import { configureStore, createSlice } from "@reduxjs/toolkit";

const counterSlice = createSlice({
  name: "counter",
  initialState: { value: 0 },
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
  },
});

export const { increment, decrement } = counterSlice.actions;
export default configureStore({ reducer: { counter: counterSlice.reducer } });
```

```
// Counter.js
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { increment, decrement } from "./store";

const Counter = () => {
  const value = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();
  return (
    <div>
      <button onClick={() => dispatch(decrement())}>-</button>
      <span>{value}</span>
      <button onClick={() => dispatch(increment())}>+</button>
    </div>
  );
};
```

Zustand (Minimal State Management)

```
import create from "zustand";

const useStore = create((set) => ({
  count: 0,
  increment: () => set((state) => ({ count: state.count + 1 })),
}));

function Counter() {
  const { count, increment } = useStore();
  return <button onClick={increment}>Count: {count}</button>;
}
```

Recoil (React State Library)

```
import { atom, useRecoilState } from "recoil";

const countState = atom({ key: "count", default: 0 });

function Counter() {
  const [count, setCount] = useRecoilState(countState);
  return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;
}
```

MobX (Observable State)

```
import { makeAutoObservable } from "mobx";
import { observer } from "mobx-react-lite";
```

```
class CounterStore {
  count = 0;
  constructor() {
    makeAutoObservable(this);
  }
  increment() {
    this.count++;
  }
}
const counterStore = new CounterStore();

const Counter = observer(() => (
  <button onClick={() => counterStore.increment()}>
    Count: {counterStore.count}
  </button>
));
```

Best Practices

- Use local state for UI and ephemeral data.
- Use Context for app-wide settings (theme, locale).
- Use Redux/Zustand/MobX for complex, shared, or persistent state.
- Keep state minimal and colocated when possible.

Chapter 8: Styling Solutions

CSS Modules

```
/* Button.module.css */
.button {
  background: #007bff;
  color: white;
  padding: 8px 16px;
  border: none;
  border-radius: 4px;
}
.button.primary {
  background: #0056b3;
}
```

```
import styles from "./Button.module.css";
<button className={styles.button + " " + styles.primary}>Primary</button>;
```

Styled Components

```
import styled from "styled-components";

const Button = styled.button`
  background: #007bff;
  color: white;
  padding: 8px 16px;
  border: none;
  border-radius: 4px;
  &:hover {
    background: #0056b3;
  }
`;

<Button>Styled Button</Button>;
```

Emotion

```
/** @jsxImportSource @emotion/react */
import { css } from "@emotion/react";

const buttonStyle = css`
  background: #007bff;
  color: white;
  padding: 8px 16px;
  border: none;
  border-radius: 4px;
`;

<button css={buttonStyle}>Emotion Button</button>;
```

Tailwind CSS

```
<button className="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-700">
  Tailwind Button
</button>
```

Sass/SCSS

```
$primary: #007bff;
.button {
  background: $primary;
  color: white;
  padding: 8px 16px;
  border: none;
  border-radius: 4px;
  &:hover {
```

```
    background: darken($primary, 10%);
  }
}
```

Utility-First CSS

- Use utility classes for rapid prototyping and consistent design (e.g., Tailwind, Bootstrap utilities).

Best Practices

- Scope styles to components.
- Prefer CSS-in-JS or modules for large apps.
- Use variables and theming for consistency.

Chapter 9: Build Tools & Bundlers

npm & Yarn

```
npm install react react-dom
# or
yarn add react react-dom
```

Webpack

```
// webpack.config.js
module.exports = {
  entry: "./src/index.js",
  output: { filename: "bundle.js", path: __dirname + "/dist" },
  module: {
    rules: [
      { test: /\.jsx?$/, use: "babel-loader", exclude: /node_modules/ },
      { test: /\.css$/, use: ["style-loader", "css-loader"] },
    ],
  },
  devServer: { static: "./dist", hot: true },
};
```

Babel

```
// .babelrc
{
  "presets": ["@babel/preset-env", "@babel/preset-react"]
}
```

Vite

```
npm create vite@latest my-app -- --template react
cd my-app
npm install
npm run dev
```

Next.js CLI

```
npx create-next-app@latest my-next-app
cd my-next-app
npm run dev
```

ESLint & Prettier

```
npm install eslint prettier --save-dev
npx eslint --init
```

Husky & Lint-Staged

```
npx husky-init && npm install
npm install lint-staged --save-dev
```

Best Practices

- Use fast dev servers (Vite, Next.js, Parcel).
- Automate linting and formatting.
- Use environment variables for config.

Chapter 10: Testing Strategies

Unit Testing (Jest)

```
// sum.js
export function sum(a, b) {
  return a + b;
}

// sum.test.js
import { sum } from "./sum";
test("adds 1 + 2 to equal 3", () => {
```

```
  expect(sum(1, 2)).toBe(3);
});
```

React Testing Library

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

test("increments counter", () => {
  render(<Counter />);
  fireEvent.click(screen.getByText("+"));
  expect(screen.getByText("1")).toBeInTheDocument();
});
```

Cypress (E2E Testing)

```
// cypress/integration/spec.js
it("loads the home page", () => {
  cy.visit("/");
  cy.contains("Welcome");
});
```

Storybook (UI Testing)

```
npx storybook init
npm run storybook
```

Best Practices

- Write tests for logic and UI.
- Use mocks for API calls.
- Automate tests in CI/CD.

Chapter 11: Performance Optimization

Code Splitting

```
import React, { lazy, Suspense } from "react";
const LazyComponent = lazy(() => import("./LazyComponent"));

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

```
      <LazyComponent />
    </Suspense>
  );
}
```

Memoization

```
import React, { useMemo } from "react";
const Expensive = ({ items }) => {
  const total = useMemo(() => items.reduce((a, b) => a + b, 0), [items]);
  return <div>Total: {total}</div>;
};
```

Virtualization

```
import { FixedSizeList as List } from "react-window";
<List height={400} itemCount={1000} itemSize={35} width={300}>
  {({ index, style }) => <div style={style}>Row {index}</div>}
</List>;
```

Image Optimization

- Use `next/image` in Next.js.
- Use modern formats (WebP, AVIF).
- Lazy load images.

Lighthouse Audits

- Use Google Lighthouse for performance checks.

Best Practices

- Minimize bundle size.
- Avoid unnecessary re-renders.
- Optimize assets and code.

Chapter 12: Development Tools

VS Code Extensions

- Prettier, ESLint, GitLens, Tailwind CSS IntelliSense, Bracket Pair Colorizer

Browser DevTools

- Inspect elements, debug JS, monitor network, performance profiling

Git & GitHub

```
git init
git add .
git commit -m "Initial commit"
git remote add origin <repo-url>
git push -u origin main
```

Postman & API Clients

- Test REST and GraphQL APIs

Figma & Design Tools

- UI/UX design, prototyping, handoff

Best Practices

- Use source control for all projects.
- Automate repetitive tasks.
- Use design systems for consistency.

Chapter 13: Deployment & DevOps

Vercel (Next.js, React)

```
npm install -g vercel
vercel
```

Netlify

```
npm install -g netlify-cli
netlify deploy
```

GitHub Pages

```
git push origin main
git subtree push --prefix dist origin gh-pages
```

Docker

```

```

```
# Dockerfile
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
CMD ["npm", "start"]
```

CI/CD (GitHub Actions)

```
# .github/workflows/ci.yml
name: CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Install
        run: npm install
      - name: Build
        run: npm run build
      - name: Test
        run: npm test
```

Environment Variables

- Use `.env` files for secrets and config.

Best Practices

- Automate deployments.
- Use environment variables for config.
- Monitor and rollback on failure.

Chapter 14: Advanced Patterns

Compound Components

```
const Tabs = ({ children }) => {
  /* ... */
};
const Tab = ({ children }) => {
  /* ... */
};
const TabPanel = ({ children }) => {
```

```
  /* ... */
};
// See React Patterns above for full example
```

Render Props

```
const Mouse = ({ render }) => {
  const [pos, setPos] = useState({ x: 0, y: 0 });
  useEffect(() => {
    const handler = (e) => setPos({ x: e.clientX, y: e.clientY });
    window.addEventListener("mousemove", handler);
    return () => window.removeEventListener("mousemove", handler);
  }, []);
  return render(pos);
};
```

Higher-Order Components (HOC)

```
function withLogger(Component) {
  return function Wrapper(props) {
    useEffect(() => {
      console.log("Mounted");
    }, []);
    return <Component {...props} />;
  };
}
```

Custom Hooks

```
function useFetch(url) {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch(url)
      .then((res) => res.json())
      .then(setData);
  }, [url]);
  return data;
}
```

Controlled vs Uncontrolled Components

```
// Controlled
<input value={value} onChange={e => setValue(e.target.value)} />
```

```
// Uncontrolled
<input ref={inputRef} />
```

Portals

```
import { createPortal } from "react-dom";
createPortal(<Modal />, document.body);
```

Error Boundaries

```
class ErrorBoundary extends React.Component {
  state = { hasError: false };
  static getDerivedStateFromError() {
    return { hasError: true };
  }
  componentDidCatch(error, info) {
    /* log error */
  }
  render() {
    if (this.state.hasError) return <h1>Something went wrong.</h1>;
    return this.props.children;
  }
}
```

Chapter 15: Popular Libraries & Packages

UI Libraries

- Material-UI (MUI)
- Ant Design
- Chakra UI
- Bootstrap
- Semantic UI

State Management

- Redux, Zustand, MobX, Recoil, Jotai

Data Fetching

- React Query, SWR, Axios

Form Libraries

- Formik, React Hook Form

Animation

- Framer Motion, React Spring, GSAP

Testing

- Jest, React Testing Library, Cypress, Storybook

Utilities

- Lodash, date-fns, classnames, uuid

Dev Tools

- ESLint, Prettier, Husky, Lint-Staged

Best Practices

- Use well-maintained libraries.
- Keep dependencies up to date.
- Prefer libraries with good documentation and community support.