

Chapter 6: React UI Development – Professional Level

6.1 UI/UX Fundamentals

6.1.1 Introduction

- ★ **UI (User Interface)** is the visual part of the application with which users interact.
- ★ **UX (User Experience)** is the overall experience a user has while using the application.
- ★ Good UI/UX makes the application **easy to use, attractive** and **user-friendly**.
- ★ In React development, building a **great UI/UX** is as important as writing clean and efficient code.

Key Points :

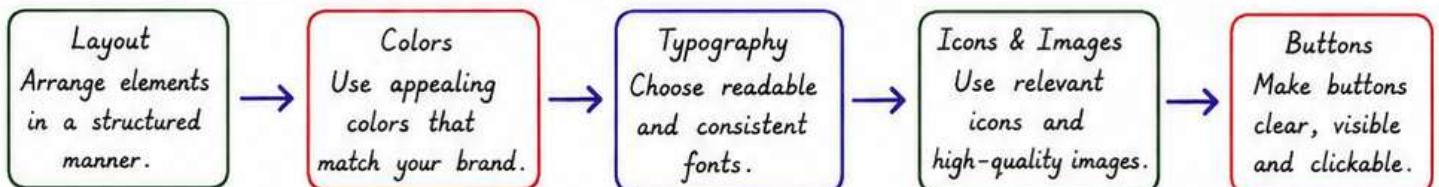
- ✓ UI focuses on how the application looks.
- ✓ UX focuses on how the application works.
- ✓ Both UI and UX together create a successful product.
- ✓ React helps in building reusable, interactive and dynamic UIs.



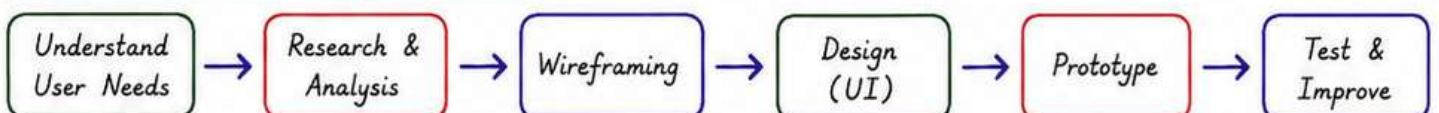
6.1.2 Principles of Good UI/UX Design

- ★ **Simplicity** – Keep the design simple and clean.
- ★ **Consistency** – Maintain uniformity in colors, fonts and layout.
- ★ **Clarity** – Information should be easy to understand.
- ★ **Feedback** – Provide immediate feedback for user actions.
- ★ **Accessibility** – Design for all users including differently-abled users.

6.1.3 Elements of UI Design



6.1.4 UI/UX Design Process



Key Takeaway :



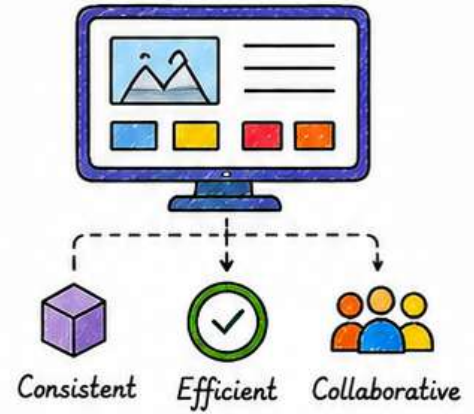
A good UI/UX increases **user satisfaction, engagement** and **trust**.
Always **design for users**, keep improving and test regularly.



6.2 Design Systems

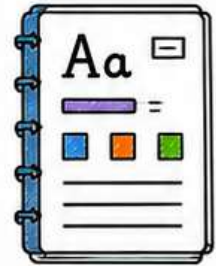
6.2.1 Introduction

- ★ A **Design System** is a collection of **reusable components**, **patterns**, **guidelines** and standards used to build **consistent** and **scalable** user interfaces.
- ★ It helps teams build products **faster** and maintain a **uniform** look and feel across the application.
- ★ In React, design systems are usually built using **reusable components** and documented for all developers.

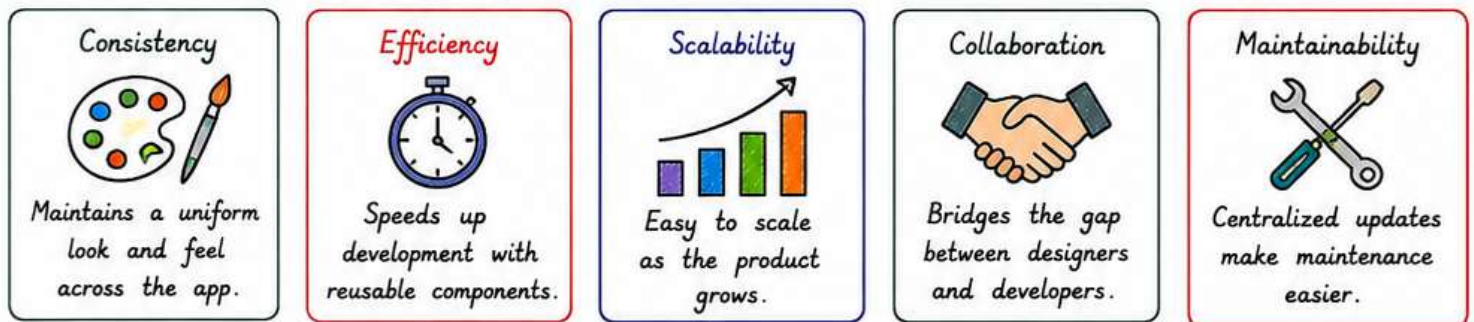


Key Points :

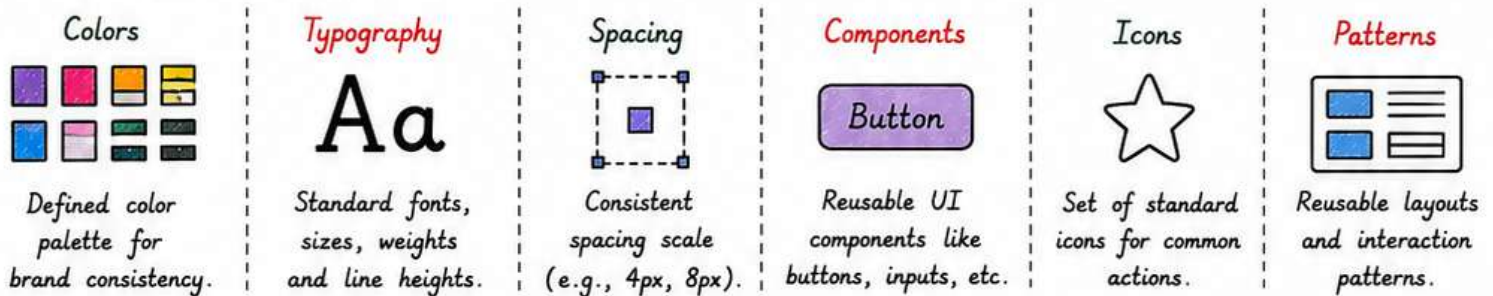
- ✓ Design systems ensure consistency across products.
- ✓ They reduce duplication and improve productivity.
- ✓ They improve collaboration between designers and developers.
- ✓ They provide a single source of truth for UI components and guidelines.



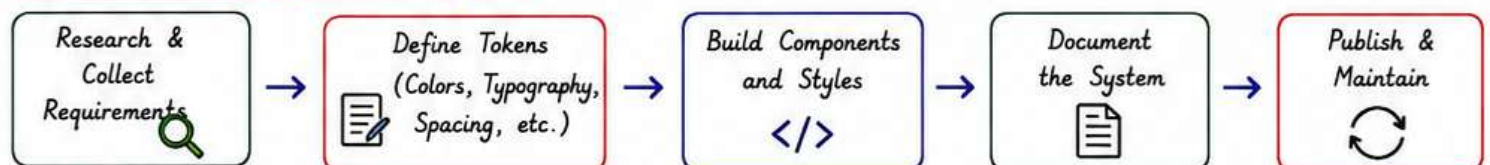
6.2.2 Benefits of Design Systems



6.2.3 Key Building Blocks



6.2.4 Design System Workflow



Key Takeaway :



A strong design system makes your React applications **consistent**, **scalable** and **easier to maintain**. It saves time, reduces errors and improves the overall **user experience**.



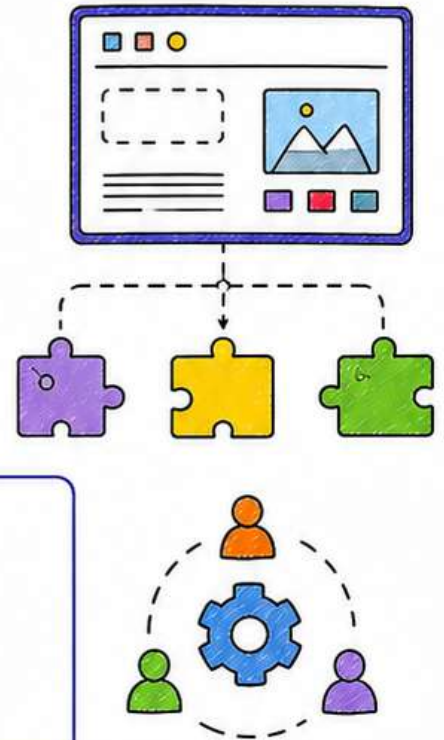
6.3 Component-Based UI Design

6.3.1 Introduction

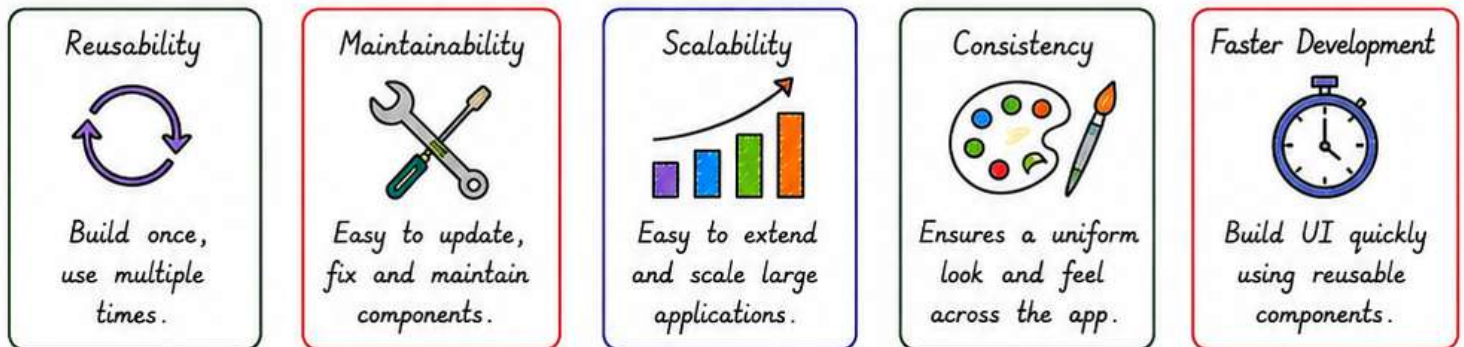
- ★ Component-based design is a way of building UIs by breaking them into **smaller, reusable pieces** called **components**.
- ★ Each component manages its own **logic, structure and style**.
- ★ Components can be **combined** to build complex and interactive user interfaces.
- ★ React follows a **component-based architecture**.

Key Points :

- ✓ Components make the code modular, reusable and maintainable.
- ✓ Each component represents a part of the UI.
- ✓ Components help in faster development and easy testing.
- ✓ They improve collaboration among developers.

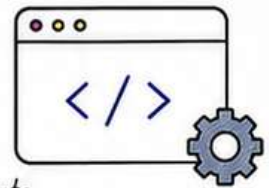


6.3.2 Benefits of Component-Based Design

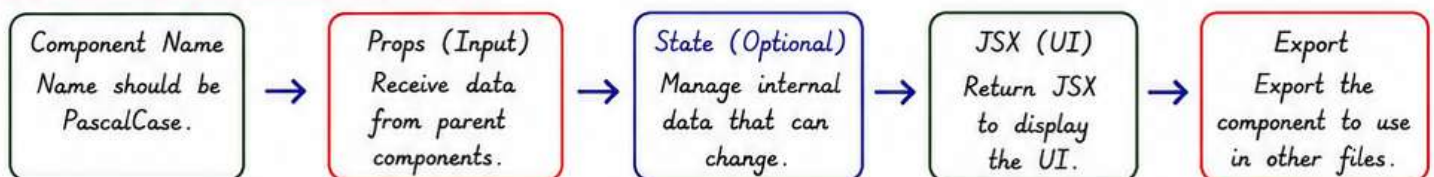


6.3.3 Types of Components in React

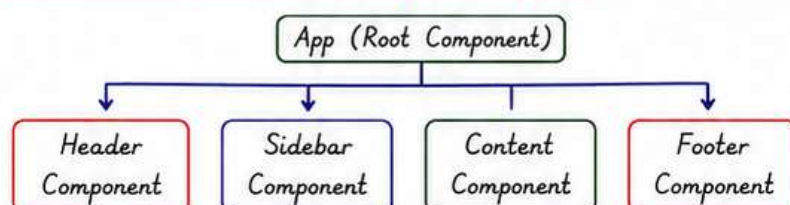
- ★ **Functional Components** - Simple JavaScript functions that return JSX.
- ★ **Class Components** - ES6 classes that extend `React.Component`.
- ★ **Presentational Components** - Handle how things look (UI).
- ★ **Container Components** - Handle logic and data, pass props to presentational components.



6.3.4 Component Structure



6.3.5 Component Hierarchy (Example)



Key Takeaway :



Component-based design is the core of React UI development. It helps in building **modular, scalable, reusable** and **easy** to maintain applications.

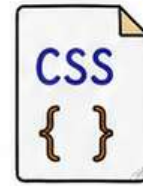
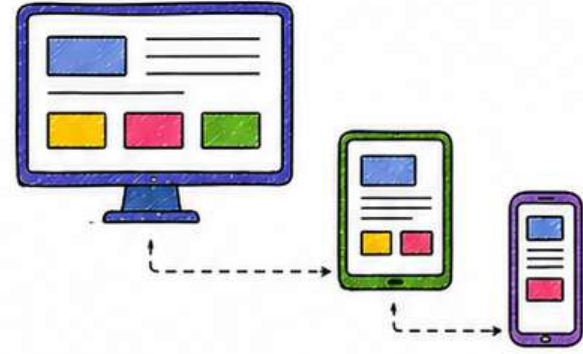
6.4 Responsive React UI

6.4.1 Introduction

- ★ Responsive design ensures that the UI looks and works well on **all devices** like desktops, tablets and mobile phones.
- ★ In React, we build responsive UIs by using **flexible layouts**, **media queries** and **responsive components**.
- ★ It improves **user experience**, accessibility and engagement.
- ★ React with CSS frameworks or CSS-in-JS makes it easy to create responsive interfaces.

Key Points :

- ✓ Responsive UI adapts to different screen sizes.
- ✓ It uses flexible grids, images and media queries.
- ✓ Mobile-first approach is recommended.
- ✓ Tools like Bootstrap, Tailwind CSS and Material UI help in building responsive designs quickly.

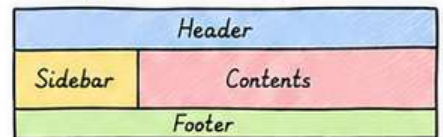


6.4.2 Common Breakpoints

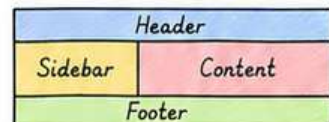
Device	Screen Width	Type
Extra Large (XL)	$\geq 1200\text{px}$	Large desktops
Large (LG)	992px - 1199px	Desktops / Laptops
Medium (MD)	768px - 991px	Tablets
Small (SM)	576px - 767px	Large phones
Extra Small (XS)	$< 576\text{px}$	Small phones

Responsive Layout Example

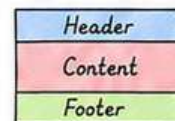
Desktop View ($\geq 1200\text{px}$)



Tablet View (768px - 991px)

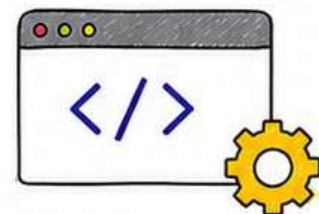


Mobile View ($< 576\text{px}$)

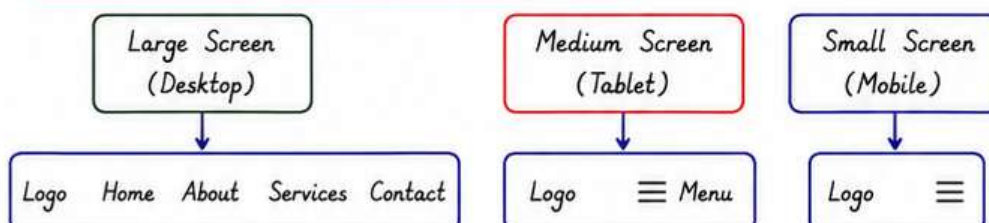


6.4.3 Techniques for Responsive UI in React

- ★ **CSS Media Queries** - Apply different styles based on screen size.
- ★ **Flexbox** - Create flexible and responsive layouts.
- ★ **Grid System** - Use CSS Grid or frameworks like Bootstrap/Tailwind.
- ★ **Responsive Images** - Use 100% width images and srcSet property.
- ★ **Mobile-First Approach** - Design for mobile first, then scale up.
- ★ **Reusable Components** - Build components that adapt to screen size.



6.4.4 Example: Responsive Navbar



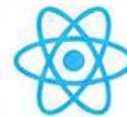
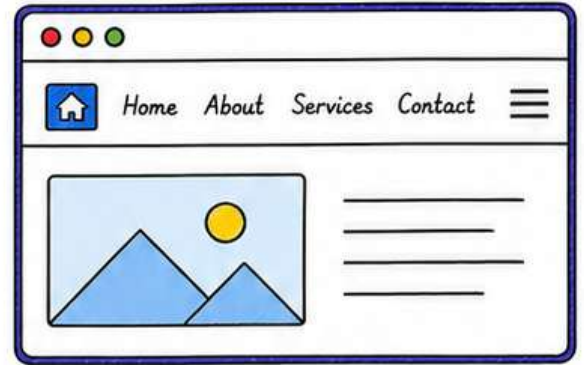
Key Takeaway :

- 💡 Responsive React UI makes your application look great on **every device**, improves usability and keeps users **engaged**.

6.5 Navigation Bars

6.5.1 Introduction

- ★ A **Navigation Bar (Navbar)** helps users **move between** different pages or sections of the application.
- ★ It is one of the **most important components** of any UI.
- ★ In React, we can build navigation bars using **components**, **React Router**, and **CSS frameworks**.
- ★ Good navigation improves **user experience** and makes the application easy to use.



Key Points :

- ✓ Navigation bars help users to navigate easily.
- ✓ They improve the structure and usability of the application.
- ✓ React Router is used to handle navigation between pages.
- ✓ Navigation bars should be responsive and mobile-friendly.



6.5.2 Types of Navigation Bars

Horizontal Navbar

Links are placed horizontally in a row.

Home About Services Contact

Vertical Sidebar

Links are placed vertically on the side.

Home
About
Services
Contact

Top Navbar with Dropdown

Menu items have dropdown options for sub-items.

Services ▼
Web Design
Development

Hamburger Menu

Used in mobile view to show/hide the menu.



6.5.3 Building Navbar in React (Using React Router)

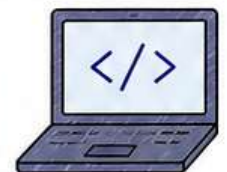
Install React Router

Create Navbar Component

Add Links using <Link>

Style the Navbar with CSS

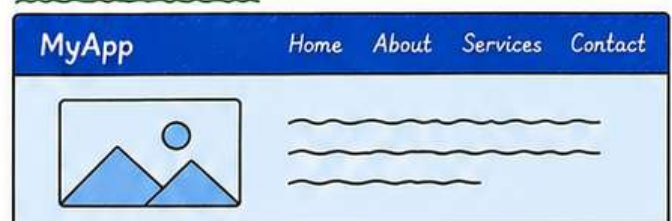
Use Navbar in App Component



6.5.4 Example: Simple Navbar Code

```
import { Link } from 'react-router-dom';  
function Navbar() {  
  return (  
    <nav className="navbar">  
      <div className="logo">MyApp</div>  
      <ul className="nav-links">  
        <li><Link to="/">Home</Link></li>  
        <li><Link to="/about">About</Link></li>  
        <li><Link to="/services">Services</Link></li>  
        <li><Link to="/contact">Contact</Link></li>  
      </ul>  
    </nav>  
  );  
}
```

Navbar Preview



Key Takeaway :

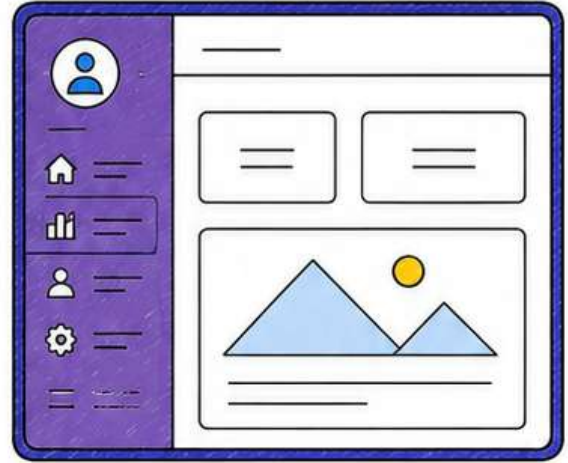


A well-designed navigation bar helps users navigate easily and improves the overall **user experience** of the application.

6.6 Sidebars

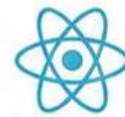
6.6.1 Introduction

- ★ A **Sidebar** is a vertical navigation menu that appears on the side of the screen.
- ★ It is used to **navigate between different sections** or pages of an application.
- ★ Sidebars are commonly used in **dashboards, admin panels** and web applications.
- ★ In React, we can build sidebars using **components, CSS frameworks** and state management.



Key Points :

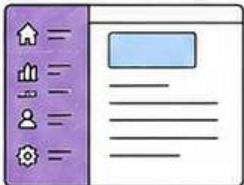
- ✓ Sidebars help organize and manage navigation in large applications.
- ✓ They save space and keep the UI clean and structured.
- ✓ They improve user experience and make navigation easy.
- ✓ Sidebars can be collapsible, responsive and interactive.



6.6.2 Types of Sidebars

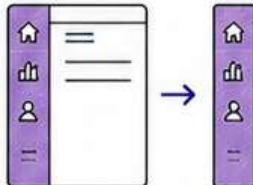
Static Sidebar

Always visible on the screen.
Used in dashboards.



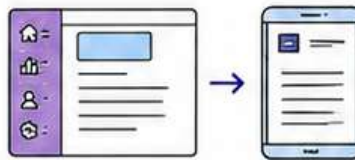
Collapsible Sidebar

Can be expanded or collapsed to save space.



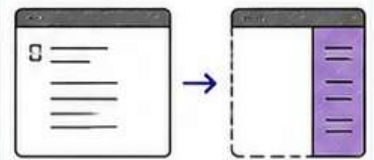
Responsive Sidebar

Visible on large screens, hidden on small screens (replaced by drawer or hamburger menu).



Off-Canvas / Drawer

Slides in from the left or right when needed, commonly in mobile view.



6.6.3 Building Sidebar in React

Install React Router (if needed)

Create Sidebar Component

Add Navigation Links

Style the Sidebar with CSS/Tailwind

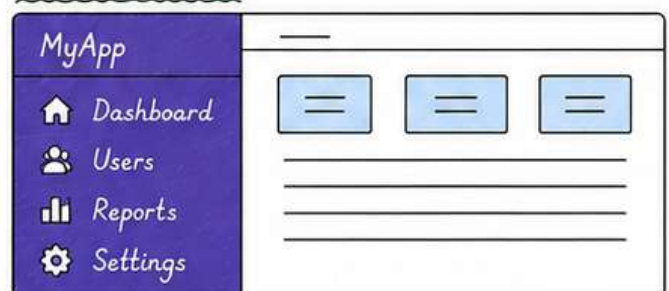
Use Sidebar in Layout or App Component



6.6.4 Example: Simple Sidebar Structure

```
function Sidebar() {  
  return (  
    <div className="sidebar">  
      <div className="logo">MyApp</div>  
      <ul className="nav-links">  
        <li><Link to="/"><img alt="home icon" data-bbox="380 860 395 875"/> Dashboard</Link></li>  
        <li><Link to="/users"><img alt="users icon" data-bbox="380 875 395 890"/> Users</Link></li>  
        <li><Link to="/reports"><img alt="reports icon" data-bbox="380 890 395 905"/> Reports</Link></li>  
        <li><Link to="/settings"><img alt="settings icon" data-bbox="380 905 395 920"/> Settings</Link></li>  
      </ul>  
    </div>  
  );  
}
```

Sidebar Preview



Key Takeaway :

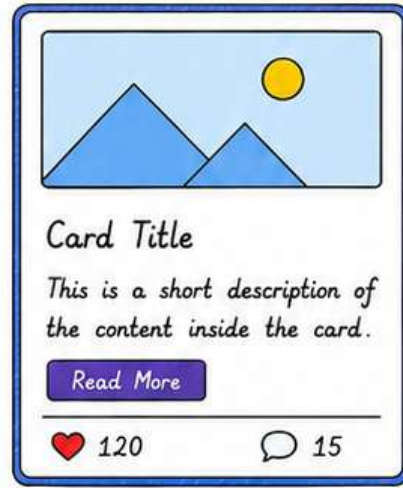


Sidebars help **structure** your application navigation, save space and provide a **better user experience**.

6.7 Cards

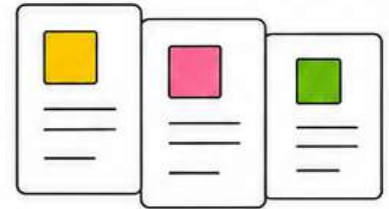
6.7.1 Introduction

- ★ A **Card** is a container that holds content like text, images, buttons and other elements.
- ★ Cards are used to display information in a **clear** and organized way.
- ★ They are commonly used in **dashboards**, **e-commerce**, **blogs**, **profiles** and many web applications.
- ★ In React, we can create cards using **components**, **CSS frameworks** or UI libraries.

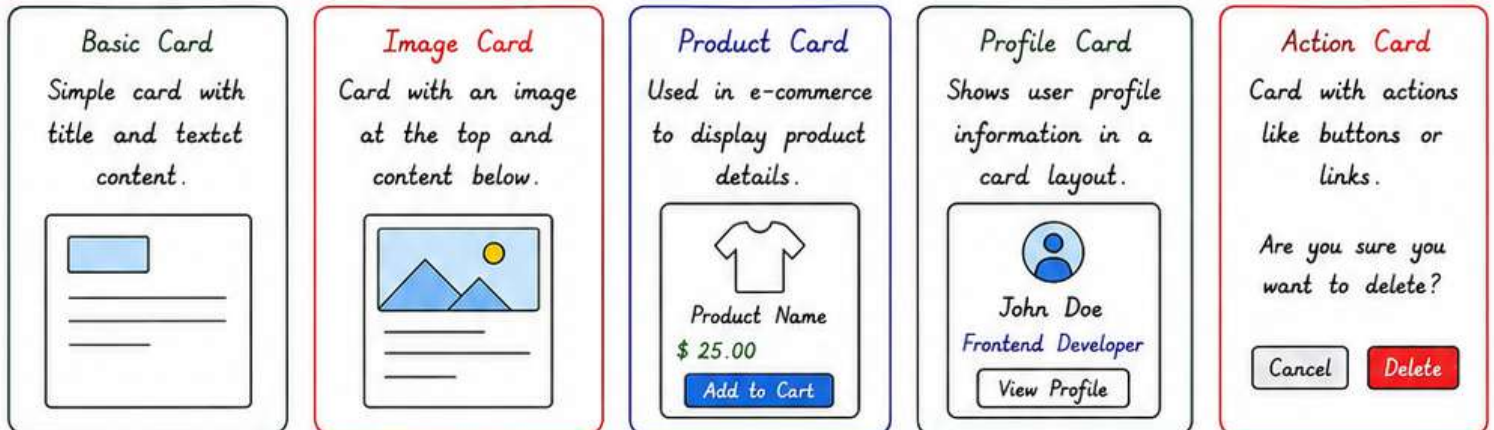


Key Points :

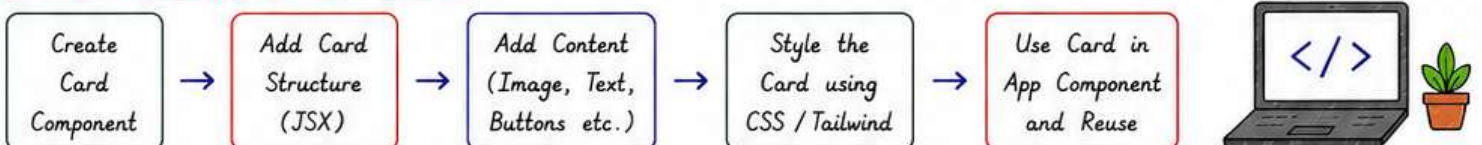
- ✓ Cards help in presenting information in a visually appealing way.
- ✓ They improve readability and user experience.
- ✓ Cards make UI look clean, modern and well-structured.
- ✓ They are responsive and can be arranged in grids or lists.



6.7.2 Types of Cards



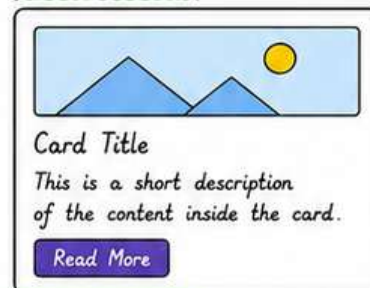
6.7.3 Building Card in React



6.7.4 Example : Simple Card Code

```
function Card() {
  return (
    <div className="card">
      
      <div className="card-content">
        <h3 className="card-title">Card Title</h3>
        <p className="card-desc">This is a short description of the content inside the card.</p>
        <button className="card-btn">Read More</button>
      </div>
    </div>
  );
}
```

Card Preview



Key Takeaway :

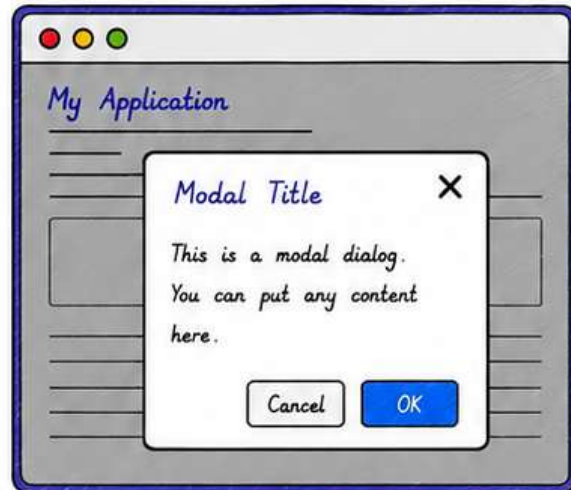


Cards are flexible UI components that help in displaying information in a **clean**, **organized** and **attractive** way.

6.8 Modals

6.8.1 Introduction

- ★ A Modal is a dialog box/popup window that appears **on top** of the current page content.
- ★ Modals are used to display **important information, forms, confirmations or alerts**.
- ★ They block interaction with the background until the modal is closed.
- ★ In React, we can create modals using **components, state and CSS frameworks**.



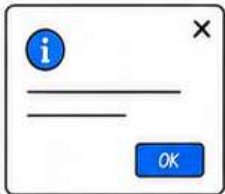
Key Points :

- ✓ Modals draw user attention to important tasks or messages.
- ✓ They improve user experience and prevent accidental actions.
- ✓ Background content is disabled until the modal is closed.
- ✓ They are commonly used in forms, alerts, confirmations, etc.

6.8.2 Types of Modals

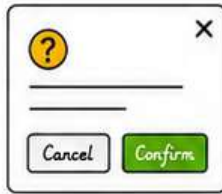
Information Modal

Displays information or messages to the user.



Confirmation Modal

Asks the user to confirm an action.



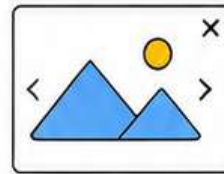
Form Modal

Used to collect data from the user in a form.



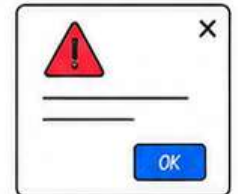
Image Modal

Displays large images, galleries or media.



Alert Modal

Shows important alerts or warnings.



6.8.3 Building Modal in React

Create Modal Component

Manage Modal Visibility using useState

Use Conditional Rendering ({isOpen &&...})

Add Content, Header, Footer and Buttons

Close Modal on Button Click or Outside Click

Common Use Cases

- User registration / login forms
- Delete confirmation
- Product quick view
- Edit profile / settings
- Alerts and notifications



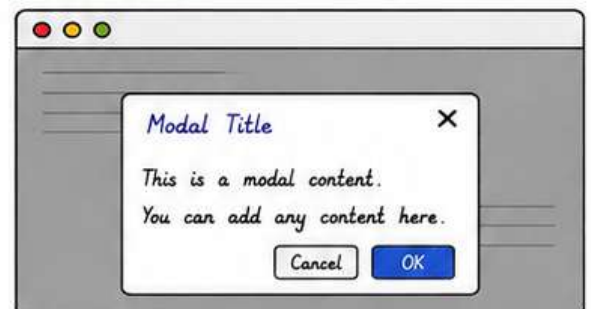
6.8.4 Example: Simple Modal Code

```
function Modal({ isOpen, onClose, children }) {  
  if (!isOpen) return null;  
  return (  
    <div className="modal-overlay" onClick={onClose}>  
      <div className="modal-content" onClick={(e) => e.stopPropagation()}>  
        <button className="close-btn" onClick={onClose}>X</button>  
        {children}  
      </div>  
    </div>  
  );  
}
```

Explanation :

- The modal is shown only when `isOpen` is true.
- Clicking on overlay or close button closes the modal.
- `stopPropagation()` prevents closing when clicking inside modal.

Modal Preview



Key Takeaway :



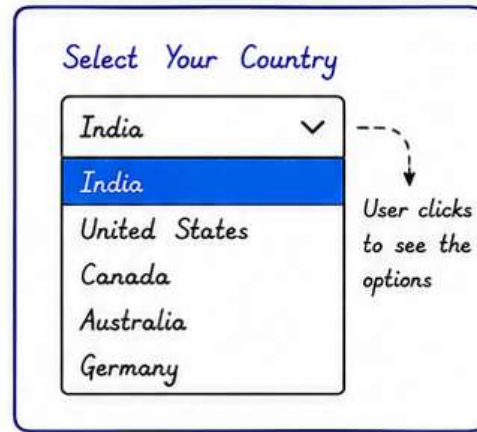
Modals are powerful UI components that help in **focusing user attention**, improving usability and handling **important actions** effectively.

6.9 Dropdowns



6.9.1 Introduction

- ★ A **Dropdown** is a UI component that allows users to **select one option** from a list of choices.
- ★ Dropdowns save space and make the UI **clean** and **user-friendly**.
- ★ They are widely used in **forms, filters, navigation menus** and **settings**.
- ★ In React, we can build dropdowns using **components, state** and **event handling**.



Key Points :

- ✓ Dropdowns help users select one option from many choices.
- ✓ They improve UI by saving space and reducing clutter.
- ✓ They are easy to use and enhance user experience.
- ✓ Dropdowns can be controlled using state in React.
- ✓ They are commonly used in forms, filters and settings.



6.9.2 Types of Dropdowns

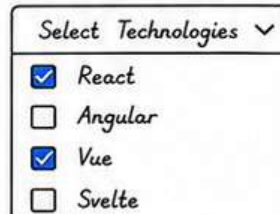
Single Select Dropdown

Allows the user to select only one option from the list.



Multi Select Dropdown

Allows the user to select multiple options from the list.



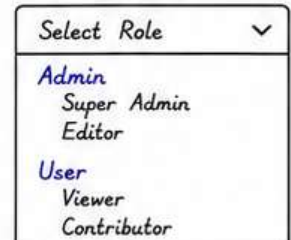
Searchable Dropdown

Allows user to search within the options.

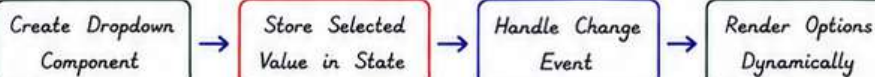


Dropdown with Groups

Options are divided into groups for better organization.



6.9.3 Building Dropdown in React



6.9.4 Example : Simple Dropdown Code

```
import { useState } from 'react';
function Dropdown() {
  const [selected, setSelected] = useState('React');
  return (
    <div>
      <label>Select Technology: </label>
      <select value={selected} onChange={(e) => setSelected(e.target.value)}>
        <option value="React">React</option>
        <option value="Angular">Angular</option>
        <option value="Vue">Vue</option>
        <option value="Svelte">Svelte</option>
      </select>
      <p>You selected: <b>{selected}</b></p>
    </div>
  );
}
```

Common Use Cases

- Selecting country, state, city in forms
- Filtering data in dashboards
- Sorting options (A to Z, Z to A)
- User role or permission selection
- Product categories in e-commerce



Dropdown Preview

Select Technology: React ▼
You selected: React

Key Takeaway :

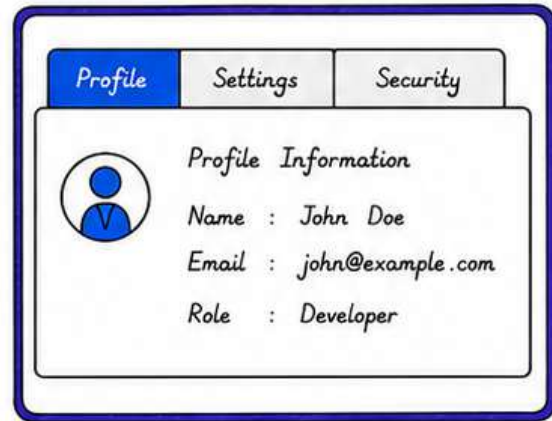


Dropdowns are simple yet powerful UI components that help users select options efficiently, save space and make the application more **user-friendly**.

6.10 Tabs

6.10.1 Introduction

- ★ A **Tab** is a UI component that allows users to switch between different content sections.
- ★ Tabs **organize related content** in the same space without changing the page.
- ★ They are widely used in **dashboards, settings pages, forms** and **admin panels**.
- ★ In React, we can build tabs using **components, state** and **conditional rendering**.



Key Points :

- ✓ Tabs help in organizing related content in a limited space.
- ✓ They improve user experience by making navigation easy.
- ✓ Only one tab content is visible at a time.
- ✓ Tabs maintain the context of the page.
- ✓ They can be styled and customized to match the design.



Tip :

Use tabs when content is related and needs quick switching.

6.10.2 Types of Tabs

Default Tabs
Standard horizontal tabs used in most applications.

Tab 1	Tab 2	Tab 3
Content for Tab 1		

Vertical Tabs
Tabs are placed vertically on the left side.

Tab 1	Content for Tab 1
Tab 2	
Tab 3	

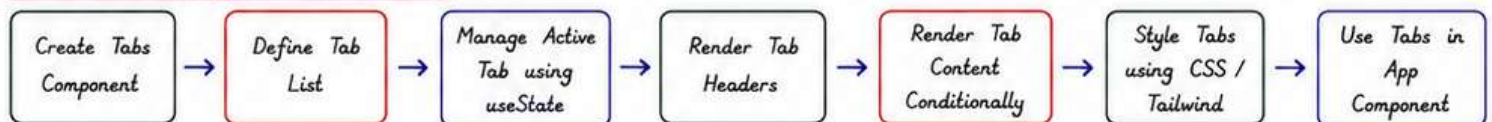
Scrollable Tabs
Used when there are many tabs and space is limited.

<	Tab 1	Tab 2	Tab 3	>
Content for Tab 1				

Icon Tabs
Tabs represented using icons instead of text.

Profile Content		

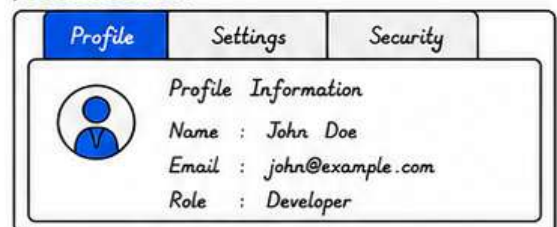
6.10.3 Building Tabs in React



6.10.4 Example: Simple Tabs Code

```
import { useState } from 'react';
function Tabs() {
  const [activeTab, setActiveTab] = useState('profile');
  return (
    <div className="tabs">
      <div className="tab-headers">
        <button className={activeTab === "profile" ? 'active': ''}
          onClick={() => setActiveTab('profile')}>Profile</button>
        <button className={activeTab === "settings" ? 'active': ''}
          onClick={() => setActiveTab('settings')}>Settings</button>
        <button className={activeTab === "security" ? 'active': ''}
          onClick={() => setActiveTab('security')}>Security</button>
      </div>
      <div className="tab-content">
        {activeTab === 'profile' && <p>Profile information content...</p>}
        {activeTab === 'settings' && <p>Settings content goes here...</p>}
        {activeTab === 'security' && <p>Security details content...</p>}
      </div>
    </div>
  );
}
```

Tabs Preview



Common Use Cases

- User profile and account settings
- Product details (Description, Reviews, FAQ)
- Admin dashboards (Users, Roles, Permissions)
- Forms with multiple sections
- Reports and analytics with multiple views



Key Takeaway :



Tabs help in displaying related content in an **organized** and **user-friendly** way by allowing quick switching between sections.

6.11 Accordions

6.11.1 Introduction

- ★ An Accordion is a UI component that shows and hides content in **expandable sections**.
- ★ It allows users to view **one section at a time** and expand it to see more details.
- ★ Accordions help in **saving space** and organizing large amounts of content.
- ★ They are commonly used in **FAQs, settings panels, documentation** and **dashboards**.
- ★ In React, we can build accordions using **components, state** and **conditional rendering**.

Key Points :

- ✓ Accordions help organize content in a clean and structured way.
- ✓ They improve readability by showing one section at a time.
- ✓ They save space and reduce page clutter.
- ✓ Users can expand or collapse sections as needed.
- ✓ They are easy to implement and enhance user experience.

6.11.2 Types of Accordions

Single Expand Accordion

Only one section can be open at a time. Opening a new section closes the previously opened one.

▼ Section 1	—
Content for section 1	
▶ Section 2	+
▶ Section 3	+

Multiple Expand Accordion

Multiple sections can be open at the same time. Users can expand or collapse any section.

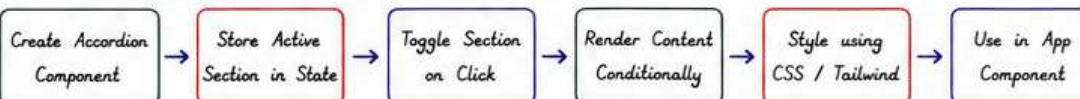
▼ Section 1	—
Content for section 1	
▼ Section 2	—
Content for section 2	
▶ Section 3	+

Default Open / Close Accordion

A section can be open by default when the page loads.

▼ Section 1 (Open by Default)	—
Content for section 1	
▶ Section 2	+
▶ Section 3	+

6.11.3 Building Accordions in React



Common Use Cases

- FAQ sections
- Settings / Preferences
- Product descriptions
- Terms and conditions
- Documentation / Guides

6.11.4 Example: Simple Accordion Code

```
import { useState } from 'react';
function Accordion() {
  const [activeIndex, setActiveIndex] = useState(0);
  const items = [
    { title: 'What is React?', content: 'React is a JavaScript library for building user interfaces.' },
    { title: 'Why use React?', content: 'It helps in building fast, scalable and dynamic UIs.' },
    { title: 'How does React work?', content: 'React uses components, state and props to build UIs.' }
  ];
  return (
    <div className="accordion">
      {items.map((item, index) => (
        <div key={index} className="item">
          <div className="header" onClick={() => setActiveIndex(activeIndex === index ? -1 : index)}>
            {item.title} <span>{activeIndex === index ? '-' : '+'}</span>
          </div>
          {activeIndex === index && (
            <div className="content">{item.content}</div>
          )}
        </div>
      ))}
    </div>
  );
};
```

Accordion Preview

▼ What is React?	—
React is a JavaScript library for building user interfaces.	
▶ Why use React?	+
▶ How does React work?	+

Key Takeaway :



Accordions are excellent UI components for organizing content. They improve **readability**, save space and provide a **clean user experience** on any device.

6.12 Breadcrumbs

6.12.1 Introduction

- ★ A Breadcrumb is a navigation aid that shows the **current location** of the user in a hierarchy of pages.
- ★ It allows users to easily **navigate back** to previous levels.
- ★ Breadcrumbs improve **usability** and help users understand the structure of the site.
- ★ They are widely used in **e-commerce** sites, **dashboards**, **blogs** and **admin panels**.
- ★ In React, we can build breadcrumbs using **components**, **routing data** and **links**.

Key Points :

- ✓ Breadcrumbs show the path from the homepage to the current page.
- ✓ They help users navigate to parent/previous pages easily.
- ✓ They improve user experience and reduce confusion.
- ✓ They also help search engines understand the site structure.
- ✓ They should be short, clean and readable.

6.12.2 Types of Breadcrumbs

Location-Based Breadcrumbs

Shows the location of the user in the hierarchy.

 [Home](#) > [Category](#) > [Product](#)

Example :

Home / Electronics / Laptops

Attribute-Based Breadcrumbs

Based on attributes or filters applied by the user.

 [Home](#) > [Brand](#) > [Apple](#)

Example :

Home / Brand / Apple

Path-Based Breadcrumbs

Shows the actual path taken by the user.

 [Home](#) > [Blog](#) > [2024](#) > [React](#)

Example :

Home / Blog / 2024 / React

History-Based Breadcrumbs

Based on the browsing history or steps followed.

 [Home](#) > [Step 1](#) > [Step 2](#)

Example :

Home / Cart / Payment

6.12.3 Building Breadcrumbs in React

Get current route path
(use `useLocation`)



Split path into segments



Build breadcrumb items with links



Render Breadcrumb component



Style using CSS / Tailwind



Use in App Component

6.12.4 Example : Simple Breadcrumb Code

```
import { Link, useLocation } from 'react-router-dom';
function Breadcrumbs() {
  const location = useLocation();
  const pathnames = location.pathname.split('/').filter(x => x);
  return (
    <nav className="text-sm text-gray-600 mb-4">
      <Link to="/" className="text-blue-600 hover:underline">Home</Link>
      {pathnames.map((value, index) => {
        const to = '/' + pathnames.slice(0, index + 1).join('/');
        const label = value.charAt(0).toUpperCase() + value.slice(1);
        return (
          <span key={to}>
            <span className="mx-2"></span>
            <Link to={to} className="text-blue-600 hover:underline">
              {label}
            </Link>
          </span>
        );
      })}
    </nav>
  );
}
```

Breadcrumb Preview

[Home](#) > [Electronics](#) > [Mobile Phones](#) > [Smartphones](#)



Product : Galaxy S24
Latest smartphone with advanced features.
₹ 79,999

Common Use Cases

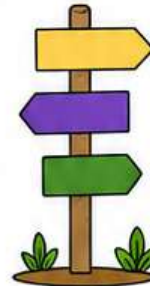
- E-commerce product navigation
- Blog categories and posts
- Admin dashboards and settings
- Documentation and knowledge bases
- Multi-step forms and wizards



Key Takeaway :



Breadcrumbs are **simple** but **powerful** navigation components that help **users** understand their location and move around the application **easily**.



Tip :

Keep breadcrumbs short. Avoid showing unnecessary levels. Show only important hierarchy.

6.13 Pagination

6.13.1 Introduction

- ★ Pagination is a UI component that divides **large amounts of data** into smaller pages.
- ★ It allows users to move between pages using **page numbers** or **navigation buttons**.
- ★ Pagination improves **performance** and **user experience** by loading only the required data.
- ★ We can build pagination in React using **components**, **state** and **event handling**.

Key Points :

- ✓ Pagination breaks data into pages for easy viewing.
- ✓ It reduces load time by fetching only required data.
- ✓ Users can navigate using page numbers or Previous / Next.
- ✓ It is essential for tables, lists, search results, and feeds.
- ✓ It improves usability and keeps the UI clean.

Example : Users List

ID	Name	Email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com
3	Alice Brown	alice@example.com
...	...	...
46	Rahul Kumar	rahul@example.com
47	Priya Singh	priya@example.com

Showing 1 to 10 of 47 users

< Prev 1 2 3 4 5 ... Next >

Tip :

Keep pagination simple.
Show only limited page numbers with Previous and Next buttons.

6.13.2 Types of Pagination

Numeric Pagination

Displays page numbers sequentially.

< Prev 1 2 3 4 5 Next >

Simple Pagination

Shows only Previous and Next buttons.

< Prev Next >

First / Last Pagination

Includes First and Last buttons for quick navigation.

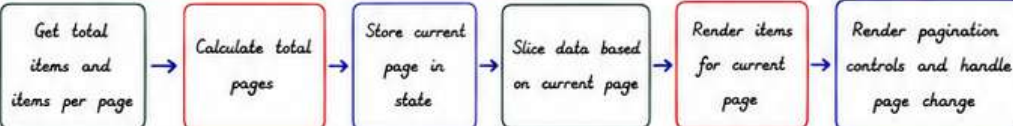
<< First < Prev 1 2 3 Next > Last >>

Ellipsis Pagination

Uses ... to hide extra page numbers.

< Prev 1 2 ... 8 9 10 Next >

6.13.3 Building Pagination in React



Common Use Cases

- Displaying large tables or lists
- Search results and filters
- Product listings in e-commerce
- Blog posts and news feeds
- User management dashboards

6.13.4 Example : Simple Pagination Code

```
import { useState } from 'react';
function Pagination({ totalItems, itemsPerPage }) {
  const [currentPage, setCurrentPage] = useState(1);
  const totalPages = Math.ceil(totalItems / itemsPerPage);
  const pages = Array.from({ length: totalPages }, (_, i) => i + 1);
  const handlePageChange = (page) => {
    if (page >= 1 && page <= totalPages) {
      setCurrentPage(page);
    }
  };
  return (
    <div className="pagination">
      <button onClick={() => handlePageChange(currentPage - 1)}
        disabled={currentPage === 1}> < Prev </button>
      {pages.map(page => (
        <button key={page}
          className={page === currentPage ? "active" : ""}
          onClick={() => handlePageChange(page)}>
            {page}
          </button>
      ))}
      <button onClick={() => handlePageChange(currentPage + 1)}
        disabled={currentPage === totalPages}> Next > </button>
    </div>
  );
}
```

Pagination Preview

ID	Name	Email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com
3	Alice Brown	alice@example.com
4	Robert Wilson	robert@example.com
5	Maria Garcia	maria@example.com

Showing 1 to 5 of 47 users

< Prev 1 2 3 4 5 ... 10 Next >

Key Takeaway :



Pagination is a powerful UI component that helps handle **large datasets** efficiently and improves application **performance** and **user experience**.

6.14 Tables

6.14.1 Introduction

- ★ A **Table** is used to display data in rows and columns in a **structured** and **organized** way.
- ★ Tables are widely used in **dashboards**, **reports**, **admin panels** and **lists**.
- ★ In React, we can create tables using **JSX**, **components** and **props**.
- ★ Tables improve **readability** and make data easy to understand.

Example : Student List

ID	Name	Branch	CGPA	Status
1	Bhavitha	CSE	9.2	Pass
2	Rohith	ECE	8.6	Pass
3	Anusha	AIML	9.0	Pass
4	Kiran	MECH	7.8	Fail

Showing 1 to 4 of 4 students

Key Points :

- ✓ Tables help in presenting data in a clean and organized format.
- ✓ They make comparison of data easy.
- ✓ Tables can be sorted, filtered and paginated.
- ✓ We can style tables using CSS, Tailwind or Bootstrap.
- ✓ They are commonly used in reports, dashboards and lists.



Tip :

Use tables only for data display, not for layout design.

6.14.2 Types of Tables

Basic Table

Simple table with rows and columns.

ID	Name	Branch
1	Alice	CSE

Striped Table

Alternate row colors for better readability.

ID	Name
1	Alice
2	Bob

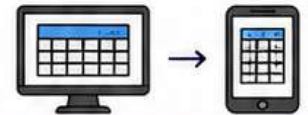
Bordered Table

Table with borders around cells.

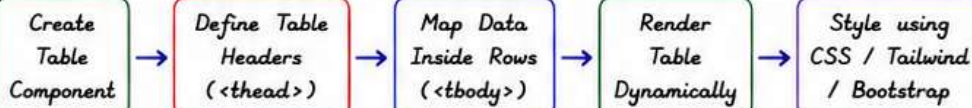
ID	Name
1	Alice
2	Bob

Responsive Table

Table adjusts according to screen size.



6.14.3 Building Table in React



Common Use Cases

- Admin dashboards
- Student / Employee lists
- Product listings in e-commerce
- Reports and analytics
- Data comparison and summaries

6.14.4 Example : Simple Table Code

```
const students = [
  { id: 1, name: "Bhavitha", branch: "CSE", cgpa: 9.2 },
  { id: 2, name: "Rohith", branch: "ECE", cgpa: 8.6 },
  { id: 3, name: "Anusha", branch: "AIML", cgpa: 9.0 }
];

function StudentTable() {
  return (
    <table className="table table-bordered">
      <thead className="bg-blue-100 text-blue-800">
        <tr>
          <th>ID</th><th>Name</th><th>Branch</th><th>CGPA</th>
        </tr>
      </thead>
      <tbody>
        {students.map(s => (
          <tr key={s.id}>
            <td>{s.id}</td>
            <td>{s.name}</td>
            <td>{s.branch}</td>
            <td>{s.cgpa}</td>
          </tr>
        ))}
      </tbody>
    </table>
  );
}
```

Table Preview

ID	Name	Branch	CGPA
1	Bhavitha	CSE	9.2
2	Rohith	ECE	8.6
3	Anusha	AIML	9.0

Showing 1 to 3 of 3 students

Key Takeaway :



Tables are essential UI components for displaying data in an **organized**, **readable** and **user-friendly** manner.

6.15 Dashboards

6.15.1 Introduction

- ★ A Dashboard is a collection of components that displays **key metrics and insights** in one place.
- ★ It helps users **monitor data, track performance** and make informed decisions.
- ★ Dashboards are widely used in **admin panels, analytics, business intelligence** and **reporting systems**.
- ★ In React, we can build dashboards using **components, charts, grids** and **layouts**.

Key Points :

- ✓ Dashboards provide a quick overview of important data.
- ✓ They help in monitoring KPIs and business performance.
- ✓ They combine multiple components like cards, charts, tables and grids.
- ✓ They improve decision making and save time.
- ✓ They can be customized for different roles and devices.

6.15.2 Components Usually Used in Dashboards

Sidebar / Navigation

For navigation between pages.



Summary Cards

To show key metrics and KPIs.



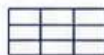
Charts

To visualize data like trends and distributions.



Tables

To display detailed data in rows and columns.



Graphs / Progress

To show progress or comparisons.



Notifications

To show alerts and updates.



6.15.3 Building a Dashboard in React

Plan Dashboard
(Define KPIs)



Create Layout
(Header, Sidebar,
Content Area)



Add Summary
Cards



Add Charts /
Graphs



Add Tables
or Lists



Add Filters /
Search



Style and
Make Responsive

6.15.4 Example : Simple Dashboard Code

```
import { useState } from 'react';
function Dashboard() {
  const [stats] = useState([
    { title: 'Total Sales', value: '₹ 12,45,000', change: '+15.2%', color: 'green' },
    { title: 'Total Users', value: '1,245', change: '+8.5%', color: 'blue' },
    { title: 'Orders', value: '890', change: '-4.3%', color: 'red' },
    { title: 'Revenue', value: '₹ 8,90,000', change: '+12.1%', color: 'purple' }
  ]);
  return (
    <div className="dashboard">
      <aside className="sidebar">
        <h2>Dashboard</h2>
        <ul>
          <li>Overview</li>
          <li>Users</li>
          <li>Sales</li>
          <li>Products</li>
          <li>Reports</li>
          <li>Settings</li>
        </ul>
      </aside>
      <main className="content">
        <div className="cards">
          {stats.map((s, i) => (
            <div className="card" key={i} style={{ borderLeft: `5px solid ${s.color}` }}>
              <h3>{s.title}</h3>
              <p className="value">{s.value}</p>
              <p className="change" style={{ color: s.color }}>{s.change} vs last month</p>
            </div>
          ))}
        </div>
        <p className="more">(Charts, Tables and other components here)</p>
      </main>
    </div>
  );
}
```

Dashboard Preview



Common Use Cases

- Admin dashboards (users, sales, orders)
- Business intelligence dashboards
- Analytics and reporting systems
- E-commerce dashboards
- Financial dashboards and monitoring



Key Takeaway :

Dashboards turn raw data into **meaningful insights**. They help businesses track performance, identify trends and make **data-driven decisions** easily.

6.16 Forms

6.16.1 Introduction

- ★ A Form is a collection of input elements that allows users to **enter and submit** data.
- ★ Forms are used in **login pages, sign up pages, contact forms, search forms** and many more.
- ★ In React, we handle forms using **state, events** and **handlers**.
- ★ React supports both **Controlled** and **Uncontrolled** components in forms.

Key Points :

- ✓ Forms collect data from users.
- ✓ We can control input values using state.
- ✓ We use **onChange** event to update state.
- ✓ We use **onSubmit** event to handle form submission.
- ✓ Forms can be validated before submission.
- ✓ Forms improve user interaction and data collection.

Example : Contact Us Form

 Name

 Email

 Phone

 Message

Gender ☐ Male ☒ Female ☐ Other

Subscribe ☒ Yes, send me updates

Tip :

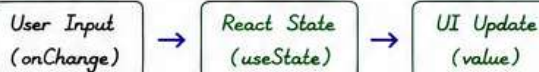
Always validate user input before submitting the form. Show clear error messages for better user experience.



6.16.2 Types of Forms in React

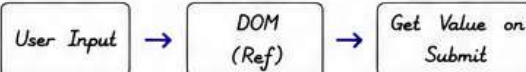
Controlled Components

Form data is handled by React state. The input value is controlled by React.

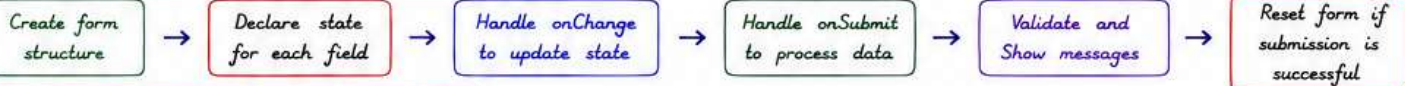


Uncontrolled Components

Form data is handled by DOM using useRef. React does not control the input value.



6.16.3 Building a Form in React (Controlled)



6.16.4 Example : Simple Form Code

```
import { useState } from 'react';
function ContactForm() {
  const [formData, setFormData] = useState({
    name: '', email: '', phone: '', message: '', gender: 'Female',
    subscribe: false
  });
  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    setFormData({ ...formData, [name]: type === 'checkbox' ? checked : value });
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    alert('Form Submitted Successfully!\n' + JSON.stringify(formData, null, 2));
  };
  return (
    <form className="form" onSubmit={handleSubmit}>
      <input type="text" name="name" placeholder="Name" value={formData.name}
        onChange={handleChange} required /><br /><br />
      <input type="text" name="email" placeholder="Email" value={formData.email}
        onChange={handleChange} required /><br /><br />
      <input type="text" name="phone" placeholder="Phone" value={formData.phone}
        onChange={handleChange} required /><br /><br />
      <textarea name="message" placeholder="Message" value={formData.message}
        onChange={handleChange} required /><br /><br />
      <label>
        <input type="radio" name="gender" value="Male" checked={formData.gender === 'Male'}
          onChange={handleChange} /> Male
        <input type="radio" name="gender" value="Female" checked={formData.gender === 'Female'}
          onChange={handleChange} /> Female
        <input type="radio" name="gender" value="Other" checked={formData.gender === 'Other'}
          onChange={handleChange} /> Other<br /><br />
      </label>
      <label>
        <input type="checkbox" name="subscribe" checked={formData.subscribe}
          onChange={handleChange} /> Yes, send me updates
      </label><br /><br />
      <button type="submit"> Submit </button>
      <button type="reset"> Reset </button>
    </form>
  );
}
export default ContactForm;
```

Form Preview

Name

Email

Phone

Message

Gender ☐ Male ☒ Female ☐ Other

Subscribe ☒ Yes, send me updates

Common Use Cases

- Login and Registration forms
- Contact us / Feedback forms
- Search and Filter forms
- Survey and Quiz forms
- Checkout and Payment forms



Key Takeaway :

- Forms are essential for **user interaction**.
- Controlled components give more **control and validation**, leading to better and reliable applications.



6.17 Loading States

6.17.1 Introduction

- ★ Loading states are used to indicate that data or content is **being fetched** or **processed**.
- ★ They improve user experience by giving **feedback** instead of showing empty content.
- ★ In React, we can manage loading states using **state variables** and **conditional rendering**.

Example : User List with Loading State

1. Loading...



Loading users...

2. Success

Users		
1	John Doe	john@example.com
2	Jane Smith	jane@example.com
3	Alice Brown	alice@example.com

Showing 1 to 3 of 3 users

3. Error



Failed to load users.

Retry

Key Points :

- ✓ Show a loader while data is being fetched.
- ✓ Show data when request is successful.
- ✓ Show an error message when request fails.
- ✓ Disable actions while loading to prevent duplicate requests.
- ✓ Loading states improve UX and reduce confusion.

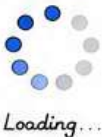
Tip :

Use spinners, skeletons, progress bars or shimmer effects as loading indicators. Always handle loading, success and error states.



6.17.2 Types of Loading Indicators

Spinner



Loading...

Progress Bar

Loading...



60%

Skeleton Loader



Shimmer Effect



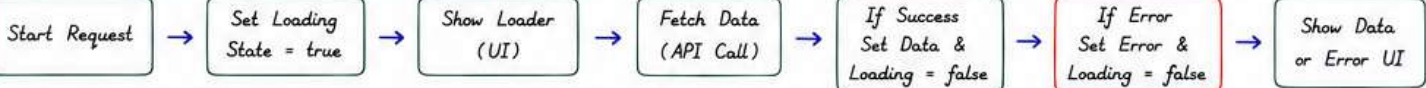
Loading...

Dots / Pulsing



Loading...

6.17.3 Building Loading States in React



6.17.4 Example : User List with Loading, Success & Error States

```
import { useState, useEffect } from 'react';
function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(res => res.json())
      .then(data => {
        setUsers(data);
        setLoading(false);
      })
      .catch(err => {
        setError('Failed to load users. ');
        setLoading(false);
      });
  }, []);
  if (loading) {
    return (
      <div className="loader">
        <div className="spinner"></div>
        <p>Loading users...</p>
      </div>
    );
  }
  if (error) {
    return (
      <div className="error">
        <p>{error}</p>
        <button onClick={() => window.location.reload()}>Retry</button>
      </div>
    );
  }
  return (
    <div>
      <h2>Users</h2>
      <table className="table">
        <thead>
          <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
          </tr>
        </thead>
        <tbody>
          {users.map(u => (
            <tr key={u.id}>
              <td>{u.id}</td>
              <td>{u.name}</td>
              <td>{u.email}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}
export default UserList;
```

Output Preview

1. Loading State



Loading users...

2. Success State

Users

ID	Name	Email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com
3	Alice Brown	alice@example.com

Showing 1 to 3 of 3 users

3. Error State



Failed to load users.

Retry

Common Use Cases

- Fetching data from APIs
- File uploads
- Processing payments
- Form submissions
- Image / video loading
- Authentication checks
- Search results



Key Takeaway :

Loading states are essential for building **responsive** and **user-friendly** applications. Always handle loading, success and error states properly. This builds **trust** and **improves UX**.

6.18 Error States

6.18.1 Introduction

- ★ Error states inform users that something **went wrong**.
- ★ They help users **understand** the problem and **how to fix it**.
- ★ Good error messages **improve** **trust** and user experience.
- ★ In React, we can show error states using **conditional rendering** and **state management**.

Example : Login Form with Error States

1. Default	2. Error State	3. Success State
Email user@example.com	Email user@example.com ✗ Enter a valid email address.	Email user@example.com ✓
Password 	Password ... ✗ Password must be at least 6 characters.	Password ✓
<button>Login</button> Welcome! Please login.	<button>Login</button> Login failed. Please try again.	✓ Login successful! Redirecting...

Key Points :

- ✓ Use **clear** and **specific** error messages.
- ✓ Show errors near the related input field.
- ✓ Use **icons** and **colors** to make errors noticeable.
- ✓ Provide **suggestions** on how to fix the error.
- ✓ Do not use technical jargon in error messages.
- ✓ Handle all possible errors gracefully.

Tip :

Error messages should be friendly, helpful and actionable.
Always focus on solving the user's problem.



6.18.2 Types of Error States

Input Validation Errors



Invalid or missing input from user.

Example : Invalid email, empty field.

Server / API Errors



Errors from server while fetching data.

Example : 500 Internal Server Error.

Network Errors



No internet or network connection issues.

Example : Network unreachable.

Permission Errors



User is not authorized to perform an action.

Example : Access denied.

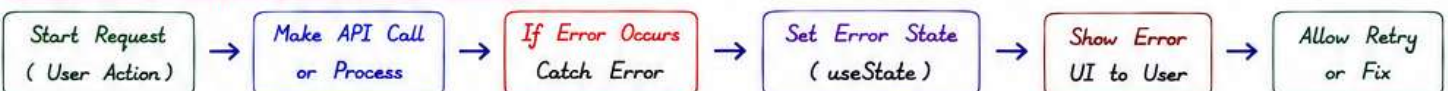
Not Found Errors

404

Resource or page not found.

Example : 404 Page Not Found.

6.18.3 Building Error States in React



6.18.4 Example : Simple Error Handling in React

```
import { useState, useEffect } from 'react';
function UserList() {
  const [users, setUsers] = useState([]);
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(res => {
        if (!res.ok) throw new Error('Failed to fetch users');
        return res.json();
      })
      .then(data => {
        setUsers(data);
        setLoading(false);
      })
      .catch(err => {
        setError(err.message);
        setLoading(false);
      });
  }, []);

  if (loading) return <p className="Loading">Loading users...</p>;
  if (error) return (
    <div className="error-box">
      <p className="error-msg">⚠️ {error}</p>
      <button onClick={() => window.location.reload()}>Retry</button>
    </div>
  );
  return (
    <div>
      <h2>Users</h2>
      <ul>
        {users.map(u => (
          <li key={u.id}>{u.name} - {u.email}</li>
        ))}
      </ul>
    </div>
  );
}
export default UserList;
```

Output Preview

1. Loading State



Loading users...

2. Error State

⚠️ Failed to fetch users

Retry

3. Success State

Users

ID	Name	Email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com
3	Alice Brown	alice@example.com

Showing 1 to 3 of 3 users

Common Use Cases

- Form validation
- Login / Authentication
- Fetching data from APIs
- File uploads
- Payment failures
- Search and filters
- Network connectivity issues

Key Takeaway :



Error states turn bad experiences into **helpful guidance**.

Clear messages, suggestions and retry options build **better** and more **reliable** applications.

6.19 Empty States

6.19.1 Introduction

- ★ Empty states are shown when there is **no data** or **content** to display.
- ★ They **guide users** on what to do next and **improve user experience**.
- ★ In React, we can create empty states using **conditional rendering** based on data availability.

Example : Empty States in Different Scenarios

1. No Data



No files found.
Upload your files
to get started.

Upload Files

2. No Messages



No messages yet.
Start a conversation
with your friends.

New Message

3. No Orders



You haven't placed
any orders yet.
Start shopping now!

Shop Now

4. No Search Results



No results found for
"React Tutorial".
Try a different keyword.

Clear Search

Key Points :

- ✓ Show empty states when there is no data.
- ✓ Provide a helpful message and action.
- ✓ Use icons and illustrations to make it user-friendly.
- ✓ Guide users on what to do next.
- ✓ Improves user experience and reduces confusion.

Tip :

A good empty state is
better than showing
a blank screen.
Always give a direction!



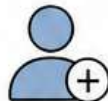
6.19.2 Types of Empty States

No Data



No data available
to display.

First Time User



New users who
haven't created
any content yet.

No Search Results



No results found for
the search query.

No Activity



No activities
or notifications.

No Items Added



No items added
yet.

6.19.3 Building Empty States in React

Check Data
Availability



If Data is Empty
Show Empty State



Display Message,
Icon & Action



Else Show
Actual Data



Allow User to
Take Action

6.19.4 Example : Simple Empty State in React

```
import React from 'react';
function UserList({ users }) {
  if (users.length === 0) {
    return (
      <div className="empty-state">
        
        <h3>No Users Found</h3>
        <p>It looks like there are no users yet.  
Add users to get started.</p>
        <button className="btn-primary">Add User</button>
      </div>
    );
  }
  return (
    <ul className="user-list">
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
export default UserList;
```

Output Preview

1. Empty State (No Users)



No Users Found

It looks like there are no users yet.
Add users to get started.

Add User

2. With Users

Users

ID	Name
1	John Doe
2	Jane Smith
3	Alice Brown

Showing 1 to 3 of 3 users

Common Use Cases

- Displaying lists with no data.
- New user onboarding.
- Search with no results.
- No notifications or activities.
- Empty cart or wishlist.
- Reports or dashboards with no data.



Key Takeaway :

Empty states turn empty
screens into **meaningful**
experiences by guiding
users and encouraging
them to take action.
They **improve usability**,
engagement and **retention**.



6.20 Toast Notifications

6.20.1 Introduction

- ★ Toast notifications are small, temporary pop-up messages that provide **quick feedback** to the user.
- ★ They appear over the content **without blocking** the main UI.
- ★ Toasts are used for **success**, **error**, **info** and **warning** messages.

Key Points :

- ✓ Toasts show short messages.
- ✓ Do not block the main UI.
- ✓ Auto-hide after a few seconds.
- ✓ Used for user feedback.
- ✓ Improve user experience.

Example : Toast Notification Types



Success Toast



Error Toast



Info Toast



Warning Toast

Tip :

Keep toast messages short, clear and useful.



6.20.2 Types of Toast Notifications

Success Toast



Indicates successful action.

Color: Green

Error Toast



Indicates an error or failure.

Color: Red

Info Toast



Provides informational messages.

Color: Blue

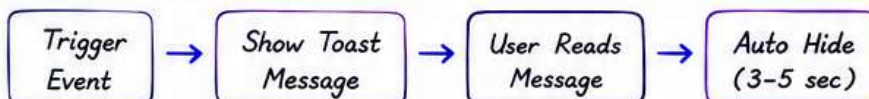
Warning Toast



Warns about possible issues.

Color: Yellow

6.20.3 Toast Workflow



6.20.4 Example : Toast Notification Code

```
import { toast } from 'react-toastify';

function App() {
  const showSuccess = () => {
    toast.success("Data saved successfully!");
  };
  const showError = () => {
    toast.error("Something went wrong!");
  };
  return (
    <div className="p-4">
      <button onClick={showSuccess} className="btn btn-success">
        Save Data
      </button>
      <button onClick={showError} className="btn btn-danger">
        Delete Data
      </button>
    </div>
  );
}

export default App;
```

Output Preview



Best Practices :

- ✓ Keep toast messages short (1-3 lines).
- ✓ Use consistent colors for message types.
- ✓ Auto-dismiss toasts for better UX.
- ✓ Do not show too many toasts at once.