Setup React Project

Create a React app using Vite or CRA.

Folder structure setup: components, assets, pages, etc.

## ✅ 1. Using Vite (Recommended – Fast & Modern)

### Step 1: Create a React project with Vite

npm create vite@latest my-react-app --template reactcd my-react-app

npm install

Alternatively with Yarn:

yarn create vite my-react-app --template reactcd my-react-app

yarn

### Step 2: Run the development server

npm run dev

## ✅ 2. Using Create React App (CRA)

### Step 1: Create a React project with CRA

npx create-react-app my-react-appcd my-react-app

### Step 2: Start the development server

npm start

## 📁 Folder Structure Setup

A scalable project should have a clean, modular folder structure. Here's a recommended layout:

perl

CopyEdit

my-react-app/

│

├── public/ # Static files like index.html, favicon, images

│

├── src/ # Source code lives here

│ ├── assets/ # Images, fonts, icons, etc.

│ │ └── logo.png

│ │

│ ├── components/ # Reusable components (Navbar, Button, etc.)

│ │ └── Navbar.jsx

│ │

│ ├── pages/ # Page-level components (Home, About, Contact)

│ │ ├── Home.jsx

│ │ └── About.jsx

│ │

│ ├── layouts/ # Common layout wrappers (e.g., MainLayout)

│ │ └── MainLayout.jsx

│ │

│ ├── routes/ # React Router config or route definitions

│ │ └── AppRoutes.jsx

│ │

│ ├── hooks/ # Custom React hooks

│ │ └── useFetch.js

│ │

│ ├── contexts/ # Context API providers

│ │ └── AuthContext.jsx

│ │

│ ├── utils/ # Utility functions or helper modules

│ │ └── formatDate.js

│ │

│ ├── App.jsx # Main app component

│ ├── main.jsx # Vite entry point (or index.js for CRA)

│ └── index.css # Global styles

│

├── .gitignore

├── package.json

└── README.md

## 🧠 What Each Folder/Component Does

assets/: Static files you want to import inside components like images or fonts.

components/: Reusable UI elements that are used across multiple pages (e.g., Button, Navbar).

pages/: Full-page components that represent routes (Home, About, etc.).

layouts/: Wrappers like MainLayout that can be used to apply a consistent structure (e.g., Navbar + Footer).

routes/: Organize your route definitions in one place.

hooks/: Custom hooks like useAuth, useToggle, etc.

contexts/: Store and manage global state using React’s Context API.

utils/: General helper functions or constants.

## 🛠 Example of React Router Setup (src/routes/AppRoutes.jsx)

jsx

CopyEdit

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';import Home from '../pages/Home';import About from '../pages/About';import MainLayout from '../layouts/MainLayout';

const AppRoutes = () => (

<Router>

<MainLayout>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</MainLayout>

</Router>

);

export default AppRoutes;

And in App.jsx:

jsx

CopyEdit

import AppRoutes from './routes/AppRoutes';

function App() {

return <AppRoutes />;

}

export default App;

## How to Import Assets

import logo from '../assets/logo.png';

const Navbar = () => (

<nav>

<img src={logo} alt="Logo" />

</nav>

);

JSX Basics

Create components using JSX.

Practice rendering HTML elements.

## 🧠 What is JSX?

JSX allows you to write HTML-like code inside JavaScript. It’s **not HTML**, but a syntax extension that gets **transformed into JavaScript** under the hood.

For example:

const element = <h1>Hello, world!</h1>;

gets compiled to:

const element = React.createElement('h1', null, 'Hello, world!');

## ✅ Rules of JSX

**Return only one parent element**: JSX must be wrapped in a single parent. Use <div>, <section>, or a fragment <> </> if needed.

return (

<div>

<h1>Title</h1>

<p>Description</p>

</div>

);

**Use** className **instead of** class: Since class is a reserved JavaScript word, JSX uses className.

**Use camelCase for attributes**: For example: onClick, tabIndex, maxLength.

**JSX expressions use** {}: To insert dynamic JavaScript in JSX, wrap it in {}.

## 🧱 Creating Basic Components Using JSX

### 1. Functional Component (Recommended)

function Welcome() {

return <h1>Welcome to React!</h1>;

}

Or using arrow function syntax:

const Welcome = () => <h1>Welcome to React!</h1>;

You can render it in App.jsx like this:

import Welcome from './components/Welcome';

function App() {

return (

<div>

<Welcome />

</div>

);

}

## 🧪 Practice: Rendering HTML Elements with JSX

### Example 1: Basic HTML structure

function ProfileCard() {

return (

<div className="profile-card">

<h2>Jane Doe</h2>

<p>Web Developer</p>

<a href="https://github.com/janedoe" target="\_blank">GitHub</a>

</div>

);

}

### Example 2: Using JavaScript expressions

const user = {

name: 'Alice',

age: 25,

};

function UserInfo() {

return (

<div>

<h2>User Info</h2>

<p>Name: {user.name}</p>

<p>Age: {user.age}</p>

</div>

);

}

## 🎯 JSX Fragment Example

When you don’t want extra wrapping elements:

function TwoLines() {

return (

<>

<p>Line 1</p>

<p>Line 2</p>

</>

);

}

## 🧑‍💻 Practice Exercise

Try creating the following JSX component yourself:

function ProductCard() {

const product = {

name: "Wireless Mouse",

price: "$29.99",

inStock: true

};

return (

<div className="product-card">

<h2>{product.name}</h2>

<p>Price: {product.price}</p>

<p>Status: {product.inStock ? "In Stock" : "Out of Stock"}</p>

</div>

);

}

## 📦 Where to Save These Components?

Create a file like ProductCard.jsx inside your src/components/ folder:

// src/components/ProductCard.jsxexport default function ProductCard() {

return (

<div>

{/\* JSX goes here \*/}

</div>

);

}

Then in App.jsx:

import ProductCard from './components/ProductCard';

function App() {

return (

<div>

<ProductCard />

</div>

);

}

Functional Components

Create a few simple reusable components like Button, Card, etc.

Pass props to components.

## What are Functional Components?

Functional components are **JavaScript functions** that return JSX. They can accept **props** as arguments to make them reusable.

## 🧩 What are Props?

**Props (short for properties)** are like parameters passed to components. They let you pass **data** from a parent to a child component.

### Syntax:

function MyComponent(props) {

return <p>{props.message}</p>;

}

Or using **destructuring**:

function MyComponent({ message }) {

return <p>{message}</p>;

}

## ✅ Example 1: Reusable Button Component

### 📁 File: Button.jsx

function Button({ label, onClick, color }) {

return (

<button

onClick={onClick}

style={{ backgroundColor: color, padding: '10px', border: 'none', color: 'white', borderRadius: '5px' }}

>

{label}

</button>

);

}

export default Button;

### 📁 In App.jsx:

import Button from './components/Button';

function App() {

const handleClick = () => {

alert("Button clicked!");

};

return (

<div>

<Button label="Click Me" onClick={handleClick} color="blue" />

<Button label="Delete" onClick={() => alert('Deleted')} color="red" />

</div>

);

}

## ✅ Example 2: Reusable Card Component

### 📁 File: Card.jsx

function Card({ title, description, imageUrl }) {

return (

<div style={{ border: '1px solid #ccc', padding: '16px', width: '250px', borderRadius: '8px' }}>

<img src={imageUrl} alt={title} style={{ width: '100%', borderRadius: '6px' }} />

<h3>{title}</h3>

<p>{description}</p>

</div>

);

}

export default Card;

### 📁 In App.jsx:

import Card from './components/Card';

function App() {

return (

<div style={{ display: 'flex', gap: '20px' }}>

<Card

title="React"

description="A JavaScript library for building user interfaces"

imageUrl="https://reactjs.org/logo-og.png"

/>

<Card

title="Vue"

description="The Progressive JavaScript Framework"

imageUrl="https://vuejs.org/images/logo.png"

/>

</div>

);

}

## 💡 Tip: Combine Components

You can use components inside other components for **composability**.

function App() {

return (

<Card

title="React"

description="Learn once, write anywhere."

imageUrl="https://reactjs.org/logo-og.png"

>

<Button label="Learn More" onClick={() => alert("Learning React")} color="green" />

</Card>

);

}

(You’d need to modify Card.jsx to support children — I can show that next if you're interested!)

## 🧪 Practice Challenge

Create a UserCard component with these props:

name

email

avatar

onMessage (function triggered on button click)

Then render it like:

<UserCard

name="John Doe"

email="john@example.com"

avatar="https://i.pravatar.cc/150?img=3"

onMessage={() => alert('Message sent to John')}

/>

### 📁 File: UserCard.jsx

// src/components/UserCard.jsxfunction UserCard({ name, email, avatar, onMessage }) {

return (

<div style={{

border: '1px solid #ccc',

padding: '16px',

width: '250px',

borderRadius: '10px',

textAlign: 'center',

boxShadow: '0 2px 8px rgba(0,0,0,0.1)'

}}>

<img

src={avatar}

alt={name}

style={{ width: '80px', height: '80px', borderRadius: '50%' }}

/>

<h2 style={{ margin: '12px 0 6px 0' }}>{name}</h2>

<p style={{ color: '#666', marginBottom: '12px' }}>{email}</p>

<button

onClick={onMessage}

style={{

backgroundColor: '#007bff',

color: '#fff',

padding: '8px 16px',

border: 'none',

borderRadius: '6px',

cursor: 'pointer'

}}

>

Message

</button>

</div>

);

}

export default UserCard;

### 📁 In App.jsx

import UserCard from './components/UserCard';

function App() {

const handleMessage = () => {

alert('Message sent to John');

};

return (

<div style={{ padding: '40px', display: 'flex', justifyContent: 'center' }}>

<UserCard

name="John Doe"

email="john@example.com"

avatar="https://i.pravatar.cc/150?img=3"

onMessage={handleMessage}

/>

</div>

);

}

export default App;

State & Events

Use useState to manage simple states (like counter, toggles).

Handle click, input, form submission events.

## What is useState?

useState is a **React Hook** that allows you to create and manage **state** in a functional component.

### Syntax:

const [state, setState] = useState(initialValue);

state: the current value

setState: a function to update the value

initialValue: the starting value for the state

You must import it:

import { useState } from 'react';

## ✅ Example 1: Counter App

import { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0); // initial state is 0

const increment = () => setCount(count + 1);

const decrement = () => setCount(count - 1);

const reset = () => setCount(0);

return (

<div style={{ textAlign: 'center' }}>

<h2>Counter: {count}</h2>

<button onClick={increment}>➕ Increment</button>

<button onClick={decrement}>➖ Decrement</button>

<button onClick={reset}>🔁 Reset</button>

</div>

);

}

export default Counter;

🔥 onClick={handlerFunction} attaches a click event.

## ✅ Example 2: Toggle Button

import { useState } from 'react';

function Toggle() {

const [isOn, setIsOn] = useState(true);

const toggle = () => setIsOn(!isOn);

return (

<button onClick={toggle}>

{isOn ? 'ON 🔆' : 'OFF 🌑'}

</button>

);

}

## ✅ Example 3: Controlled Input (with useState)

import { useState } from 'react';

function TextInput() {

const [name, setName] = useState('');

const handleChange = (event) => {

setName(event.target.value);

};

return (

<div>

<input

type="text"

placeholder="Enter your name"

value={name}

onChange={handleChange}

/>

<p>Hello, {name || "stranger"}!</p>

</div>

);

}

✅ Controlled input means React controls the value via state.

## ✅ Example 4: Simple Form Submission

import { useState } from 'react';

function SimpleForm() {

const [email, setEmail] = useState('');

const [submitted, setSubmitted] = useState(false);

const handleSubmit = (event) => {

event.preventDefault(); // prevents page reload

setSubmitted(true);

console.log("Submitted email:", email);

};

return (

<form onSubmit={handleSubmit}>

<input

type="email"

placeholder="Enter your email"

value={email}

onChange={(e) => setEmail(e.target.value)}

/>

<button type="submit">Submit</button>

{submitted && <p>Thanks! We'll contact you at {email}</p>}

</form>

);

}

onSubmit, onChange, and event.preventDefault() are crucial in form handling.

| **Event Type** | **JSX Attribute** | **Example Handler** |
| --- | --- | --- |
| Click | onClick | onClick={() => doSomething()} |
| Input Change | onChange | onChange={handleInput} |
| Form Submit | onSubmit | onSubmit={handleFormSubmit} |

## ✅ LoginForm Component

### 📁 File: LoginForm.jsx

import { useState } from 'react';

function LoginForm() {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const [submitted, setSubmitted] = useState(false);

const handleSubmit = (event) => {

event.preventDefault(); // Prevents page refresh

setSubmitted(true);

alert(`Username: ${username}\nPassword: ${password}`);

// You can also send the data to an API here

};

return (

<div style={{ maxWidth: '300px', margin: '0 auto' }}>

<h2>Login</h2>

<form onSubmit={handleSubmit}>

<div style={{ marginBottom: '10px' }}>

<label>Username:</label><br />

<input

type="text"

value={username}

onChange={(e) => setUsername(e.target.value)}

placeholder="Enter username"

required

/>

</div>

<div style={{ marginBottom: '10px' }}>

<label>Password:</label><br />

<input

type="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

placeholder="Enter password"

required

/>

</div>

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

</form>

{submitted && (

<p style={{ marginTop: '20px', color: 'green' }}>

Login submitted successfully!

</p>

)}

</div>

);

}

export default LoginForm;

## 🧪 Use it in App.jsx

import LoginForm from './components/LoginForm';

function App() {

return (

<div style={{ padding: '50px' }}>

<LoginForm />

</div>

);

}

export default App;

Props Drilling

Create parent-child-grandchild components passing data via props.

## 🧠 What is Props Drilling?

**Props Drilling** is when you pass data **down multiple levels** of components, even if some components in the middle don't use that data directly — they just pass it along.

### 📊 Visualization:

scss

CopyEdit

App (Parent)

└── Profile (Child)

└── UserDetails (Grandchild)

We’ll pass user data (like name and email) from App → Profile → UserDetails.

## ✅ Step-by-Step Example

### 1️⃣ App.jsx — The Parent

jsx

CopyEdit

import Profile from './components/Profile';

function App() {

const user = {

name: "Jane Doe",

email: "jane@example.com",

age: 28

};

return (

<div style={{ padding: '20px' }}>

<h1>Welcome to the User Page</h1>

<Profile user={user} />

</div>

);

}

export default App;

### 2️⃣ Profile.jsx — The Child

import UserDetails from './UserDetails';

function Profile({ user }) {

return (

<div style={{ border: '1px solid #ccc', padding: '15px', borderRadius: '8px' }}>

<h2>User Profile</h2>

{/\* Passing data down again \*/}

<UserDetails user={user} />

</div>

);

}

export default Profile;

### 3️⃣\ UserDetails.jsx — The Grandchild

jsx

CopyEdit

function UserDetails({ user }) {

return (

<div style={{ marginTop: '10px' }}>

<p><strong>Name:</strong> {user.name}</p>

<p><strong>Email:</strong> {user.email}</p>

<p><strong>Age:</strong> {user.age}</p>

</div>

);

}

export default UserDetails;

## 🔍 Summary

App holds the user data.

Profile receives it via props and passes it down to UserDetails.

UserDetails finally uses the data.

## 🎯 Goal:

Company holds data about an employee.

Department receives it and passes it down.

Employee displays the data.

## ✅ 1. App.jsx (Acts as Company)

import Department from './components/Department';

function App() {

const employee = {

name: "Rahul Sharma",

role: "Frontend Developer",

id: "EMP1023",

department: "Engineering"

};

return (

<div style={{ padding: '30px' }}>

<h1>Company Dashboard</h1>

<Department employee={employee} />

</div>

);

}

export default App;

## ✅ 2. Department.jsx (Child component)

import Employee from './Employee';

function Department({ employee }) {

return (

<div style={{ border: '1px solid #666', padding: '20px', borderRadius: '10px', marginTop: '20px' }}>

<h2>Department: {employee.department}</h2>

{/\* Passing data to grandchild \*/}

<Employee employee={employee} />

</div>

);

}

export default Department;

## ✅ 3. Employee.jsx (Grandchild component)

function Employee({ employee }) {

return (

<div style={{ backgroundColor: '#f0f0f0', padding: '15px', borderRadius: '8px', marginTop: '10px' }}>

<h3>Employee Info</h3>

<p><strong>ID:</strong> {employee.id}</p>

<p><strong>Name:</strong> {employee.name}</p>

<p><strong>Role:</strong> {employee.role}</p>

</div>

);

}

export default Employee;

## 📂 Folder Structure

src/

├── App.jsx

└── components/

├── Department.jsx

└── Employee.jsx

## 🧠 Key Concept Recap:

| **Component** | **Role** | **Uses Prop?** |
| --- | --- | --- |
| App | Holds data | Yes |
| Department | Passes prop | No (just relays it) |
| Employee | Displays data | Yes |

List & Keys

Display a list of users/products using .map().

## What is .map()?

.map() is a JavaScript method used to **transform arrays**. In React, we use it to **render a list of components** dynamically.

array.map((item, index) => {

return <Component key={index} {...item} />;

});

## 🔑 What are Keys in React?

A **key** is a special prop you must provide when rendering lists.

It helps React **identify which items changed, are added, or are removed**.

It should be a **unique and stable value** (like an ID).

## ✅ Example: Displaying a List of Users

### 1️⃣ Sample Data:

const users = [

{ id: 1, name: "Alice", email: "alice@example.com" },

{ id: 2, name: "Bob", email: "bob@example.com" },

{ id: 3, name: "Charlie", email: "charlie@example.com" },

];

### 2️⃣ UserList.jsx – Displaying the List

function UserList({ users }) {

return (

<div>

<h2>User Directory</h2>

<ul>

{users.map((user) => (

<li key={user.id}>

<strong>{user.name}</strong> — {user.email}

</li>

))}

</ul>

</div>

);

}

export default UserList;

### 3️⃣ Use in App.jsx

import UserList from './components/UserList';

function App() {

const users = [

{ id: 1, name: "Alice", email: "alice@example.com" },

{ id: 2, name: "Bob", email: "bob@example.com" },

{ id: 3, name: "Charlie", email: "charlie@example.com" },

];

return (

<div style={{ padding: '20px' }}>

<UserList users={users} />

</div>

);

}

export default App;

## 🧠 Why is the key prop important?

Imagine React re-renders this list.

If you **don’t provide a unique key**, React **won’t know which item is which**, leading to bugs or slow updates.

**NEVER** use index as a key if the list is dynamic (can be reordered or changed).

## 🛍 Example for Products?

const products = [

{ id: 'p1', name: "iPhone 15", price: 999 },

{ id: 'p2', name: "Samsung Galaxy S23", price: 899 },

{ id: 'p3', name: "Google Pixel 8", price: 799 },

];

You can do:

products.map(product => (

<ProductCard key={product.id} product={product} />

));

Conditional Rendering

Render components or elements based on state/props.

## 🔍 What is Conditional Rendering?

In React, **conditional rendering** means you display something **only if a certain condition is true**.

Think of it like regular JavaScript conditions, but inside JSX.

## ✅ Common Methods of Conditional Rendering

### 1️⃣ Using if or if...else (outside JSX)

if (isLoggedIn) {

return <Dashboard />;

} else {

return <LoginForm />;

}

### 2️⃣ Using the ****Ternary Operator**** (inside JSX)

{isLoggedIn ? <Dashboard /> : <LoginForm />}

### 3️⃣ Using ****Logical AND (****&&****)****

{isAdmin && <AdminPanel />}

## 🧠 When is it used?

Show login vs logout

Toggle visibility

Change layout based on role

Display error messages

Show loading indicators

## ✅ Example 1: Show "Welcome" or "Please Login"

function Greeting({ isLoggedIn }) {

return (

<div>

{isLoggedIn ? (

<h2>Welcome back!</h2>

) : (

<h2>Please log in to continue.</h2>

)}

</div>

);

}

Use in App.jsx:

<Greeting isLoggedIn={true} />

## ✅ Example 2: Show/Hide Details with Button

import { useState } from 'react';

function ToggleDetails() {

const [show, setShow] = useState(false);

return (

<div>

<button onClick={() => setShow(!show)}>

{show ? "Hide Details" : "Show Details"}

</button>

{show && (

<p>

Here are some awesome hidden details revealed only when you click the button!

</p>

)}

</div>

);

}

🔁 This uses && to show content when show is true.

## ✅ Example 3: Conditional Styling + Message

function StatusMessage({ status }) {

return (

<p style={{ color: status === 'error' ? 'red' : 'green' }}>

{status === 'error' ? 'Something went wrong.' : 'All systems go!'}

</p>

);

}

Use like:

<StatusMessage status="error" />

## 🎯 Summary of Techniques

| **Syntax** | **When to Use** |
| --- | --- |
| if / else | Complex multi-branch logic |
| Ternary ? : | Simple either/or rendering |
| Logical && | Show if true, hide if false |
| Optional chaining | Render only if value exists (user?.name) |