React Workshop Tutorial

Basics • Components • Props • Routes • Data Fetching

What you’ll build

In this workshop we’ll create a tiny app with a home page, an about page, and a list of Star Wars people fetched from a public API. Along the way we’ll learn core React concepts and best practices.

1. Prerequisites & Setup

You'll need Node.js (v18+) and a package manager (npm, pnpm, or yarn). We'll use Vite to scaffold a React app because it's fast and simple.

Create the project:

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

Open http://localhost:5173 in your browser. You should see the Vite + React starter.

2. React Basics

React lets you build user interfaces from small, reusable pieces called components. A component is a function that returns JSX (HTML-like syntax).

Hello World component:

```
function Hello() {
  return <h1>Hello, React!</h1>;
}

export default function App() {
  return (
    <div>
      <Hello />
    </div>
  );
}
```

JSX is transformed to JavaScript. Remember: components must start with a capital letter, and you can return only one parent element (wrap siblings with a fragment <>... or a div).

3. Components & Props

Props are inputs to components. They make components reusable and configurable. Think of props like function parameters.

A reusable Button component:

```
// src/components/Button.jsx
export default function Button({ kind = 'primary', children, onClick }) {
  const base = {
    padding: '8px 12px',
    borderRadius: 8,
    border: '1px solid #ccc',
    cursor: 'pointer'
  };
  const styles = kind === 'primary'
    ? { ...base, background: '#1f6feb', color: 'white', borderColor: '#1f6feb' }
    : { ...base, background: 'white' };
  return (
    <button style={styles} onClick={onClick}>
      {children}
    </button>
  );
}
```

Using the component:

```
import Button from './components/Button.jsx'

export default function App() {
  function handleClick() {
    alert('Clicked!');
  }
  return (
    <div>
      <Button onClick={handleClick}>Primary</Button>
      <Button kind="ghost" onClick={handleClick}>Ghost</Button>
    </div>
  );
}
```

Prop tips: Use default values to make components easier to consume; validate assumptions with TypeScript or PropTypes in larger codebases.

4. State & Events (Quick Intro)

State lets components remember information. Use the useState hook for local state. Keep state minimal and derive what you can.

```
import { useState } from 'react'

export default function Counter() {
  const [count, setCount] = useState(0)
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  )
}
```

5. Routing with React Router

Routing lets you map URLs to components. We'll use React Router for client-side routing.

```
npm install react-router-dom
```

```
// src/main.jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import { createBrowserRouter, RouterProvider } from 'react-router-dom'
import App from './App.jsx'
import Home from './routes/Home.jsx'
import About from './routes/About.jsx'
import People from './routes/People.jsx'

const router = createBrowserRouter([
  {
    path: '/',
    element: <App />,
    children: [
      { path: '/', element: <Home /> },
      { path: '/about', element: <About /> },
      { path: '/people', element: <People /> }
    ]
  }
])
```

```
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
)
```

```
// src/App.jsx
import { Link, Outlet } from 'react-router-dom'

export default function App() {
  const navStyle = { display: 'flex', gap: 12, marginBottom: 16 }
  return (
    <div>
      <nav style={navStyle}>
        <Link to="/">Home</Link>
        <Link to="/people">People</Link>
        <Link to="/about">About</Link>
      </nav>
      <Outlet />
    </div>
  )
}
```

Each child route renders inside the of the parent. Links change the URL without full page reloads.

6. Data Fetching

There are many ways to fetch data in React. We'll start with the built-in fetch API and the useEffect hook. For larger apps, consider libraries like React Query.

```
// src/routes/People.jsx
import { useEffect, useState } from 'react'

export default function People() {
  const [people, setPeople] = useState([])
  const [loading, setLoading] = useState(True)
  const [error, setError] = useState(null)

  useEffect(() => {
    async function load() {
      try {
        const res = await fetch('https://swapi.dev/api/people/?page=1')
        if (!res.ok) throw new Error(`Request failed: ${res.status}`)
        const json = await res.json()
        setPeople(json.results)
      } catch (e) {
        setError(e)
      } finally {
        setLoading(false)
      }
    }
    load()
  }, [])

  if (loading) return <p>Loading…</p>
  if (error) return <p>Something went wrong: {String(error)}</p>

  return (
    <ul>
      {people.map(p => (
        <li key={p.name}>{p.name} — Height: {p.height}</li>
      ))}
    </ul>
  )
```

```
}
```

Avoid fetching inside render; useEffect runs after the component renders. Clean up subscriptions or async tasks if the component can unmount mid-request.

Bonus: Data Fetching with React Query

React Query simplifies caching, refetching, and loading states.

```
npm install @tanstack/react-query
```

```
// src/main.jsx (wrap app with QueryClientProvider)
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
const client = new QueryClient()
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <QueryClientProvider client={client}>
      <RouterProvider router={router} />
    </QueryClientProvider>
  </React.StrictMode>
)
```

```
// src/routes/People.jsx using React Query
import { useQuery } from '@tanstack/react-query'

async function fetchPeople() {
  const res = await fetch('https://swapi.dev/api/people/?page=1')
  if (!res.ok) throw new Error('Network response was not ok')
  return res.json()
}

export default function People() {
  const { data, isLoading, error } = useQuery({ queryKey: ['people', 1], queryFn: fetchPeople })
  if (isLoading) return <p>Loading…</p>
  if (error) return <p>Error: {String(error)}</p>
  return (
    <ul>
      {data.results.map(p => <li key={p.name}>{p.name}</li>)}
    </ul>
  )
}
```

7. Project Structure & Best Practices

```
src/
  components/
    Button.jsx
  routes/
    Home.jsx
    About.jsx
    People.jsx
  App.jsx
  main.jsx
```

Keep components small and focused, lift state up when multiple components need it, and co-locate files close to where they’re used. Prefer declarative code and composition over inheritance.

8. Exercises

1 Add a search box on the People page to filter by name.

2 Create a Person details route that shows height, mass, and birth year.

3 Extract a component for loading states and reuse it.

4 Convert one component to TypeScript (rename to .tsx).

9. Wrap-up

You’ve learned React basics, components and props, simple routing, and data fetching. From here, explore forms (controlled vs. uncontrolled), context for cross-tree data, and performance tools like memo and useMemo.