Component Composition

The children prop, nesting components, and the difference between composition and inheritance -- with a reusable Card component mini project.

Beginner 12 min

Last updated: August 18, 2026
Read online: https://www.learnforgex.com/en/topics/component-composition

---

Component Composition

In Components and Props, we wrote small components and sent them data. This lesson shows how to COMBINE those small pieces into bigger UIs -- this is called "composition."

What Is the children Prop?

`children` is a special prop every component gets automatically. Whatever you write between a component's opening and closing tags becomes that component's `children`:

```
// "children" is a special prop that every component automatically has.
// Whatever you write between a component's OPENING and CLOSING tags becomes
// that content, reaching the component as props.children.
function Box({ children }) {
  return <div className="box">{children}</div>;
}

function App() {
  return (
    <Box>
      <p>This text is Box's children.</p>
    </Box>
  );
}

console.log(App());
```

When we write `<Box><p>...</p></Box>`, the `<p>...</p>` part becomes `Box`'s `children`. This works a bit differently from the regular props we saw in Props -- you don't pass `children` as an attribute, you write it INSIDE the tag.

Nested Components

Components can be nested -- a component can contain other components, which can themselves contain other components:

```
// Components can be nested -- a component can contain other components,
// which can in turn contain their own components. This is how a large UI
// is built up from small pieces.
function Avatar() {
  return <img src="/avatar.png" alt="Profile picture" />;
}

function UserName() {
  return <span>Jane Smith</span>;
}

function UserProfile() {
  return (
    <div>
      <Avatar />
      <UserName />
    </div>
  );
}

function App() {
  return <UserProfile />;
}

console.log(App());
```

`UserProfile` contains `Avatar` and `UserName`; `App` contains `UserProfile`. This is how you build a large UI out of small components that each focus on one job.

Composition vs. Inheritance (A Quick Look)

In object-oriented programming, "inheritance" means a class inherits behavior from another class. React components have NO such inheritance:

```
// "Inheritance" is when, in Object-Oriented Programming, a class inherits
// properties from another class. React components have NO such inheritance --
// instead, "composition" is used: you combine small components by placing
// them INSIDE a larger component.

// Composition -- the way this project (and React) does it:
function Card({ children }) {
  return <div className="card">{children}</div>;
}

function ProfileCard() {
  return (
    <Card>
      <h3>Jane Smith</h3>
      <p>Frontend Developer</p>
    </Card>
  );
}

// You would NOT write inheritance like "class ProfileCard extends Card { ... }"
// in React -- the React team says composition is sufficient and simpler for
// nearly every scenario.

console.log(ProfileCard());
```

In React, components are combined through composition, not inheritance -- you put small components inside a larger one using `children`. The React team has said composition is enough for nearly every scenario, which is why you won't see a pattern like extending a component with `extends` in React.

Summary and Glossary

`children` is a special prop that carries whatever content is written between a component's opening and closing tags. Components are nested to build larger UIs. In React, components are combined through composition, not inheritance.

Glossary

`children` — A special prop carrying the content written between a component's opening and closing tags.

Composition — Combining small components to build a larger UI.

Inheritance — A class inheriting behavior from another class -- a pattern React does NOT use between components.

Practical Project

There's a real, runnable example project that brings together the concepts from this category (Components, Props, Component Composition): **Components & Props Demo**.

It shows multiple components, props (with destructuring and default values), and composition via `children`, all working together. You can download it and run it yourself, and read through the code line by line:

```
git clone https://github.com/cdurgun/react-course-projects.git
cd react-course-projects
npm install
cd projects/components-props
npm run dev
```

The `react-course-projects` repo uses **npm workspaces** -- `npm install` only needs to run once, at the repo root, and every project folder shares the same dependencies (no separate `node_modules` per folder). If you've already run `npm install` at the root, you can just `cd react-course-projects/projects/components-props` and run `npm run dev`.