You're absolutely on the right track—and your code is a great example to explore how `App.jsx` fits into the bigger picture. Let's break it down and clarify the purpose of `App.jsx` using your code and structure.

🧠 What's Happening in Your Code?

You have two components being rendered inside `<StrictMode>` in `main.jsx`:

```
<StrictMode>
  <App />
  <QuizApp />
</StrictMode>
```

This means **both `App` and `QuizApp` are rendered side by side** inside the `<div id="root">` in `index.html`.

🧩 So What's the Role of `App.jsx` Here?

Right now, your `App.jsx` is very simple:

```
function App() {
  return (
    <>
      This is the App component.
    </>
  );
}

export default App;
```

This is totally valid—it's just a placeholder. But in a real-world app, `App.jsx` would typically be the **main container** that wraps or organizes other components like `QuizApp`, `Header`, `Footer`, etc.

🏗️ How You Could Use `App.jsx` More Effectively

Instead of rendering both `App` and `QuizApp` separately in `main.jsx`, you could **move `QuizApp` inside `App.jsx`** like this:

Updated `App.jsx`:

```
import { QuizApp } from './components/QuizApp.jsx';

function App() {
  return (
    <>
      <h1>Welcome to My App</h1>
      <QuizApp />
    </>
  );
}

export default App;
```

Updated `main.jsx`:

```
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
);
```

Now, `App.jsx` becomes the **central hub** of your UI. It's where you:

- Import and organize components
- Add layout and structure
- Apply styles from `App.css` (if you have one)
- Potentially manage shared state or routing

🧹 Why This Is Better

- **Cleaner `main.jsx`**: Only one component (`App`) is rendered.
- **Better organization**: You control layout and logic from one place.
- **Scalability**: You can easily add more components to `App.jsx` without touching `main.jsx`.

🧠 What About Rarely Used Components?

Great question. Components that aren't used often (like modals, popups, or admin panels) can still live in the `components` folder. You just import and render them **conditionally** in `App.jsx` or wherever they're needed.

Example:

```
{showModal && <ModalComponent />}
```

🧭 Summary

File	Role

`index.html`	Static HTML with `<div id="root">` for React to inject UI
`main.jsx`	Entry point that renders `<App />` into the DOM
`App.jsx`	Root component that organizes and renders other components
`QuizApp.jsx`	A feature component used inside `App.jsx` or directly in `main.jsx`
`components/`	Folder to keep reusable components organized