Implementing loading.tsx

In Next.js 15, explore the loading.tsx file in Next.js, which allows us to create loading states visible to users while content is loading. how to implement a basic loading state, the benefits of using loading UI, and how it enhances user experience by making your app feel responsive.

In **Next.js 15**, the `loading.tsx` file is part of the **App Router** (introduced in Next.js 13+), and it's used to create **route-level loading UI**. It provides a built-in way to show a loading indicator while the page or layout content is being fetched or rendered — improving the perceived performance of your app.

🔧 How to Implement a Basic `loading.tsx` File

1. Create the `loading.tsx` File

Inside a route or layout folder (e.g., `app/dashboard/`), create a `loading.tsx` file:

```
app/
├── dashboard/
│   ├── page.tsx
│   ├── loading.tsx ✅
```

2. Basic Example

tsx

```tsx
// app/dashboard/loading.tsx

export default function Loading() {
  return (
    <div style={{ padding: '2rem', textAlign: 'center' }}>
      <p>Loading dashboard...</p>
    </div>
```

```
  );
}
```

This component is **automatically rendered** by Next.js while the content in `page.tsx` or nested layouts is loading.

💡 When Is It Triggered?

- When using `React.lazy`, **data fetching**, or **streaming components**, this `loading.tsx` file is shown until the full content is ready.
- Works with **Server Components** and **Suspense boundaries**.

🎯 Benefits of Using a `loading.tsx` File

✅ 1. Improved User Experience

- Prevents blank screens while data is loading.
- Gives users feedback that something is happening.
- Makes your app feel faster and more responsive.

✅ 2. Built-in Suspense Support

- No need to manually wrap every component with `<Suspense>`.
- Next.js handles it based on your file structure.

✅ 3. Per-route Customization

- You can have different loading UIs for different pages:
  - `app/dashboard/loading.tsx`
  - `app/profile/loading.tsx`

🎨 Enhance Your Loading UI (Optional)

You can add spinners, skeleton loaders, or animations:

```tsx
// app/products/loading.tsx

export default function Loading() {
```

```
  return (
    <div className="spinner-container">
      <div className="spinner" />
      <p>Loading products...</p>

      <style jsx>{`
        .spinner-container {
          display: flex;
          flex-direction: column;
          align-items: center;
          padding: 2rem;
        }
        .spinner {
          width: 40px;
          height: 40px;
          border: 4px solid #ccc;
          border-top-color: #0070f3;
          border-radius: 50%;
          animation: spin 1s linear infinite;
        }
        @keyframes spin {
          to {
            transform: rotate(360deg);
          }
        }
      `}</style>
    </div>
  );
}
```

📈 Summary: Why Use `loading.tsx` in Next.js 15?

Feature	Benefit
✨ Built-in support	No extra setup — just add a file
⚡ Fast feedback	Reduces perceived load time

Feature	Benefit
🧠 Simplicity	Keeps logic out of your main page
🎨 Customizable	Tailor loading UIs for each route

Would you like an example with **data fetching**, like showing a loading state while fetching data from an API?

yes

Great! Let's go through a **complete example** in **Next.js 15** using the App Router that includes:

- A page with **data fetching**.
- A `loading.tsx` to show a loading UI while data is being fetched.
- The `page.tsx` that fetches and displays data.

🗂 Folder Structure

kotlin

```
app/
├── users/
│   ├── page.tsx      ← The main page fetching user data
│   ├── loading.tsx   ← Shown while page is loading
```

1️⃣ `loading.tsx` — Simple Loading UI

tsx

```tsx
// app/users/loading.tsx

export default function Loading() {
```

```
  return (
    <div style={{ textAlign: 'center', marginTop: '2rem' }}>
      <p>🔄 Loading users...</p>
    </div>
  );
}
```

2️⃣ `page.tsx` — Fetch Data from API

tsx

```tsx
// app/users/page.tsx

// This is a Server Component
async function fetchUsers() {
  // Simulating API call with delay
  await new Promise((res) => setTimeout(res, 2000)); // 2-second delay
  const response = await fetch('https://jsonplaceholder.typicode.com/users');
  return response.json();
}

export default async function UsersPage() {
  const users = await fetchUsers();

  return (
    <main style={{ padding: '2rem' }}>
      <h1>Users</h1>
      <ul>
        {users.map((user: any) => (
          <li key={user.id}>
            {user.name} — {user.email}
          </li>
        ))}
      </ul>
    </main>
  );
}
```

🧪 What Happens

1. User visits `/users`.
2. `page.tsx` starts fetching data.
3. During the fetch delay, `loading.tsx` is shown automatically.
4. Once data is ready, it replaces the loading screen with the user list.

✅ **No Suspense wrapping or manual state management is needed** — Next.js handles it using the file-based convention.

🧠 Tip: Want to Make It More Realistic?

Replace the fake delay with actual API loading logic, or integrate your real backend or database.

🚀 Bonus: With Suspense in Client Components

If you're using **Client Components** that fetch on the client side (e.g., with `useEffect` or React Query), you can still use Suspense boundaries and even custom fallbacks — but `loading.tsx` is best for **initial page-level data loading** in Server Components.

Would you like a version of this using **Client Components with Suspense** or **React Query** next?