Lazy Loading

- Lazy loading is a design pattern used in web development to defer the loading of non-essential resources (like images, components, or scripts) until they are needed.
- By deferring the loading of non-critical resources, developers can improve performance, save bandwidth, and create a smoother user experience.
- Whether you're working with images, components, or routes, lazy loading offers flexibility and scalability, making it a key tool in web performance optimization.
- This optimization technique improves the initial load time of a web application, enhances performance, and provides a better user experience by prioritizing critical content.

Why Lazy Loading is used

- **Improved Performance:** Reduces the amount of data downloaded during the initial load. Improves page load speed by only loading resources when they are required.
- **Bandwidth Savings:** Avoids downloading unnecessary resources that users might not interact with. Particularly useful for users on slow or limited internet connections.
- **Better User Experience:** Ensures that users can quickly access visible content while background content is loaded as needed.
- **Optimized Resource Usage:** Reduces the load on the server and the user's device by minimizing the resources loaded upfront.

Key Concepts of Lazy Loading

- **Lazy Loading Images:** Images are only loaded when they are about to appear in the user's viewport.
- **Code Splitting:** JavaScript files are divided into smaller chunks, and only the required chunk is loaded at runtime.
- **Lazy Loading Components:** React or other frameworks can dynamically load components when needed, reducing the initial bundle size.
- **Intersection Observer API:** Used to detect when elements are about to come into the viewport, triggering the loading of the resource.
- **Placeholder and Fallbacks:** Placeholder elements are displayed while lazy-loaded resources are being fetched, providing a seamless user experience.

Use Cases of Lazy Loading

- **Images in Long Lists or Feeds:** Social media feeds, e-commerce product pages, or blogs with many images benefit from lazy loading.
- **Large JavaScript Applications:** Web apps with many components or pages use lazy loading to split code and load components as needed.
- **Third-Party Libraries:** Libraries and dependencies, like analytics or map services, are loaded lazily to reduce initial overhead.
- **Videos and Media:** Large videos or background media are loaded only when the user interacts with them.
- **Infinite Scrolling:** Applications like Instagram or Twitter load content dynamically as users scroll down.

Example: Lazy Loading in React

App.js

```
import React, { Suspense, lazy } from "react";
const LazyComponent = lazy(() => import("./MyComponent"));
function App() {
  return (
    <div> <h1>Welcome to Lazy Loading Example</h1>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />      </Suspense>
    </div>
  );}
export default App;
```

MyComponent.js

```
import React from "react";
function MyComponent() {
  return (
    <div>
      <h2>This is a Test Component!</h2>
      <p>It contains a lot of content and dependencies, but thanks to lazy loading,
it is only loaded when needed.      </p>
    </div>  );}
export default MyComponent;
```

Explanation

- lazy(): Dynamically imports MyComponent only when it's rendered and loads it when it is required in the application.
- Suspense: Provides a fallback (like a loader) while the lazy-loaded component is being fetched. Suspense is used to show a fallback UI (like a loading spinner or message) while the lazy-loaded component is being fetched and rendered.
- fallback UI: The fallback prop of Suspense defines the UI to display while the LazyComponent is being loaded.

Role of Suspense and fallback in Lazy Loading

Suspense and its fallback prop play a crucial role in implementing lazy loading in React. They help manage the UI during the loading of lazy-loaded components, ensuring a seamless user experience.

Suspense

Suspense is a React component designed to handle components that load asynchronously, such as those implemented with React.lazy. It acts as a boundary, allowing React to "pause" rendering of parts of the UI until the lazy-loaded component or other asynchronous resource is ready.

Key Responsibilities of Suspense

- **Boundary for Lazy Loading:** It wraps around the lazy-loaded component and waits for the component to load before rendering it.
- **Fallback UI Management:** While the lazy-loaded component is being fetched, React displays the fallback UI defined in the fallback prop of Suspense.
- **Error Handling Integration:** Works in conjunction with ErrorBoundary to gracefully handle errors during loading.

fallback

The fallback prop of Suspense defines the UI to display while the lazy-loaded component or resource is being fetched. This can be:

- A simple loading message.
- A spinner or loader animation.
- A placeholder UI to match the app's design.

Key Responsibilities of fallback

- **User Feedback:** Provides visual feedback to the user, indicating that something is loading.
- **Placeholder Content:** Prevents a blank screen while waiting for the lazy-loaded component to render.
- **Enhances User Experience:** Makes the loading process less jarring and keeps users engaged.

Benefits of Suspense and fallback

- **Non-Blocking UI:** The rest of the UI is rendered immediately, even if the lazy-loaded component is not ready.
- **Graceful Loading:** Users see a loading indicator (defined in fallback) instead of a blank screen.
- **Improved User Experience:** Gives users immediate feedback, reducing perceived wait times.

Example: Lazy Loading Routes (React Router)

App.js

```
import React, { lazy, Suspense } from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
// Lazy load components
const Home = lazy(() => import("./Home"));
const About = lazy(() => import("./About"));
const Contact = lazy(() => import("./Contact"));
function App() {
  return (
    <Router>
     { /* Suspense wraps the routes to handle lazy loading */ }
     <Suspense fallback={<div>Loading...</div>}>
       <Routes>
         <Route path="/" element={<Home />} />
         <Route path="/about" element={<About />} />
         <Route path="/contact" element={<Contact />} />
       </Routes>
     </Suspense>
```

```
    </Router>
  );}
export default App;
```

Home.js

```
import React from "react";
function Home() {
  return (
    <div>
      <h1>Welcome to the Home Page</h1>
      <p>This is the Home component, loaded lazily!</p>    </div>
  );}
export default Home;
```

About.js

```
import React from "react";
function About() {
  return (
    <div>
      <h1>Welcome to the About us Page</h1>
      <p>This is the About us component, loaded lazily!</p>    </div>
  );}
export default About;
```

Contact.js

```
import React from "react";
function Contact() {
  return (
    <div>
      <h1>Welcome to the Contact Page</h1>
      <p>This is the Contact component, loaded lazily!</p>    </div>
  );
}
export default Contact;
```

Example: API Calling using Suspense

Products.js

```
import { useState, useEffect } from "react";
import axios from 'axios';

const Products = () => {
  const [status, setStatus] = useState("");
  const [productData, setProductData] = useState([]);

  useEffect(() => {
    getProductsData();
  }, []);

  const getProductsData = async () => {
    try {
      const response = await axios("https://dummyjson.com/products");
      if (response.status === 200) {
        setProductData(response.data);
      }
    } catch (error) {
      setStatus("error");
    }
  };

  if (status === "error") {
    return <h1>Error ...</h1>;
  }

  return (
    <>
      {productData &&
      productData?.products &&
      productData?.products.length !== 0 ? (
        productData?.products.map((item, index) => (
          <div key={index}>
            <h4>{item?.title}</h4>
            <p>{item.description}</p>
```

```
          </div>
        ))
      ) : (
        <>No Data Found</>
      )}
    </>
  );
};

export default Products
```

App.js

```
import { Suspense, lazy } from "react";

const Products = lazy(() => import("./Products"));

const App = () => {
  return (
    <>
      <h2>ReactJS Asynchronous Calls</h2>
      <Suspense fallback={<p>Loading ...</p>}>
        <Products/>
      </Suspense>
    </>
  );
};

export default App;
```

LEARN 2 EARN LABS

Example: implementing lazy loading to fetch details from the JSONPlaceholder API.

App.js

```
import React, { lazy, Suspense } from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

const PostList = lazy(() => import("./PostList"));
const PostDetails = lazy(() => import("./PostDetails"));

function App() {
  return (
    <Router>
      <h1>Lazy Loading Example with JSONPlaceholder</h1>
      <Suspense fallback={<div>Loading...</div>}>
        <Routes>
          <Route path="/" element={<PostList />} />
          <Route path="/post/:id" element={<PostDetails />} />
        </Routes>
      </Suspense>
    </Router>
  );
}

export default App;
```

PostList.js

```
import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";

function PostList() {
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then((response) => response.json())
```

```
      .then((data) => {
        setPosts(data);
        setLoading(false);
      });
  }, []);

  if (loading) return <div>Loading posts...</div>;

  return (
    <div>
      <h2>Posts</h2>
      <ul>
        {posts.map((post) => (
          <li key={post.id}>
            <Link to={`/post/${post.id}`}>{post.title}</Link>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default PostList;
```

PostDetails.js

```
import React, { useEffect, useState } from "react";
import { useParams, Link } from "react-router-dom";

function PostDetails() {
  const { id } = useParams();
  const [post, setPost] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .then((response) => response.json())
```

```
      .then((data) => {
        setPost(data);
        setLoading(false);
      });
  }, [id]);

  if (loading) return <div>Loading post details...</div>;

  return (
    <div>
      <h2>{post.title}</h2>
      <p>{post.body}</p>
      <Link to="/">Back to Posts</Link>
    </div>
  );
}

export default PostDetails;
```

LEARN 2 EARN LABS

Example : Adding Loader

App.js

```
import React, { useState, useEffect,lazy, Suspense } from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
// Lazy load components
const Home = lazy(() => import("./Home"));
const About = lazy(() => import("./About"));
const Contact = lazy(() => import("./Contact"));
// Loader component
const Loader = () => {
 return (
  <div>
   <img src="https://media.giphy.com/media/3oEjI6SIIHBdRxXI40/giphy.gif"
    alt="loader"/>
  </div>
 );
};
function App() {
 const [loading, setLoading] = useState(true);
 // Simulate an initial loader
 useEffect(() => {
  const timer = setTimeout(() => {
   setLoading(false);
  }, 2000);
  return () => clearTimeout(timer);
 }, []);
 // Display a loader initially
 if (loading) {
  return <Loader />;
 }
 return (
  <Router>
   { /* Suspense wraps the routes to handle lazy loading */ }
   <Suspense fallback={<div>Loading...</div>}>
    <Routes>
     <Route path="/" element={<Home />} />
```

```
            <Route path="/about" element={<About />} />
            <Route path="/contact" element={<Contact />} />
          </Routes>
        </Suspense>
      </Router>
    );}
export default App;
```

Home.js

```
import React from "react";
function Home() {
  return (
    <div>
      <h1>Welcome to the Home Page</h1>
      <p>This is the Home component, loaded lazily!</p>    </div>
  );}
export default Home;
```

About.js

```
import React from "react";
function About() {
  return (
    <div>
      <h1>Welcome to the About us Page</h1>
      <p>This is the About us component, loaded lazily!</p>    </div>
  );}
export default About;
```

Contact.js

```
import React from "react";
function Contact() {
  return (
    <div>
      <h1>Welcome to the Contact Page</h1>
      <p>This is the Contact component, loaded lazily!</p>    </div>
  );}
export default Contact;
```

Example: add a reusable Loader component

Loader.js

```
import React from "react";

function Loader() {
  return (
    <div style={styles.loaderContainer}>
      <div style={styles.spinner}></div>
      <p style={styles.text}>Loading...</p>
      { /* CSS for animation */ }
      <style>{`
        @keyframes spin {
          0% { transform: rotate(0deg); }
          100% { transform: rotate(360deg); }
        }
      `}</style>
    </div>
  );
}

const styles = {
  loaderContainer: {
    display: "flex",
    flexDirection: "column",
    alignItems: "center",
    justifyContent: "center",
    height: "100vh",
    backgroundColor: "#f8f8f8",
  },
  spinner: {
    width: "60px",
    height: "60px",
    border: "6px solid #ddd",
    borderTop: "6px solid #3b82f6", // Customize spinner color
    borderRadius: "50%",
    animation: "spin 1s linear infinite",
```

```
  },
  text: {
    marginTop: "15px",
    fontSize: "18px",
    color: "#555",
  },
};

export default Loader;
```

App.js

```
import React, { lazy, Suspense } from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Loader from "./Loader"; // Import the new Loader component

const PostList = lazy(() => import("./PostList"));
const PostDetails = lazy(() => import("./PostDetails"));

function App() {
  return (
    <Router>
      <h1>Lazy Loading Example with JSONPlaceholder</h1>
      { /* Use the new Loader as the fallback */ }
      <Suspense fallback={<Loader />}>
        <Routes>
          <Route path="/" element={<PostList />} />
          <Route path="/post/:id" element={<PostDetails />} />
        </Routes>
      </Suspense>
    </Router>
  );
}

export default App;
```

PostList.js

```
import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";
import Loader from "./Loader"; // Import the new Loader component

function PostList() {
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then((response) => response.json())
      .then((data) => {
        setPosts(data);
        setLoading(false);
      });
  }, []);

  // Use the new Loader
  if (loading) return <Loader />;

  return (
    <div>
      <h2>Posts</h2>
      <ul>
        {posts.map((post) => (
          <li key={post.id}>
            <Link to={`/post/${post.id}`}>{post.title}</Link>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default PostList;
```

LEARN 2 EARN LABS

PostDetails.js

```
import React, { useEffect, useState } from "react";
import { useParams, Link } from "react-router-dom";
import Loader from "./Loader"; // Import the new Loader component

function PostDetails() {
  const { id } = useParams();
  const [post, setPost] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .then((response) => response.json())
      .then((data) => {
        setPost(data);
        setLoading(false);
      });
  }, [id]);

  // Use the new Loader
  if (loading) return <Loader />;

  return (
    <div>
      <h2>{post.title}</h2>
      <p>{post.body}</p>
      <Link to="/">Back to Posts</Link>
    </div>
  );
}

export default PostDetails;
```

LEARN 2 EARN LABS

Advantages of Lazy Loading

- **Performance Optimization:** Faster initial page load times and reduced Time to Interactive (TTI).
- **Resource Efficiency:** Minimizes unnecessary resource loading, saving bandwidth and memory.
- **Better SEO (for Images):** Modern lazy loading methods are SEO-friendly, ensuring images and content are still indexed by search engines.

Disadvantages of Lazy Loading

- **SEO Concerns:** If not implemented correctly, search engine crawlers might not load lazy content.
- **Dependency on JavaScript:** Lazy loading often requires JavaScript, which might be disabled on some devices (legacy support).
- **User Experience:** If poorly implemented, users may notice a delay in loading visible content.

LEARN 2 EARN
LABS