⚡ 1. Code Splitting – Tách mã JS để tối ưu tải trang

📌 Mục tiêu:

- Tách mã JavaScript lớn thành các **chunk nhỏ**.
- Chỉ tải **một phần mã cần thiết** khi trang bắt đầu render (lazy loading các phần còn lại).

📦 Cách làm:

- Sử dụng React.lazy và Suspense để tải component theo yêu cầu:

tsx

```
const SettingsPage = React.lazy(() => import('./SettingsPage'));

<Route path="/settings" element={
  <Suspense fallback={<Loading />}>
    <SettingsPage />
  </Suspense>
} />
```

✅ Tốt nhất:

- Code splitting theo **route-level** (mỗi trang 1 chunk).
- Tránh chia quá nhỏ → tăng số lượng request HTTP → phản tác dụng.

🧠 2. Component và State Optimization

📌 Nguyên tắc:

- **Đặt state gần nơi sử dụng nhất**.
- **Tách state ra nhiều phần nhỏ** thay vì gom chung toàn app → tránh re-render không cần thiết.

- Dùng React.memo, useMemo, useCallback khi cần tránh render lại component con.

🧮 **State với giá trị khởi tạo tốn hiệu năng:**

ts

```
// ❌ tốn hiệu năng (gọi hàm mỗi lần render)
const [data, setData] = useState(expensiveComputation());

// ✅ tối ưu (chạy 1 lần duy nhất)
const [data, setData] = useState(() => expensiveComputation());
```

🧱 **Gợi ý lib:**

- Nếu state phức tạp → dùng jotai, zustand, recoil (hỗ trợ atomic update).
- Với React Context, chỉ nên dùng cho dữ liệu ít thay đổi như theme, user.

👶 **3. Children Prop – Tối ưu re-render với JSX con**

🌟 **Ý tưởng:**

- Khi truyền JSX thông qua children, nếu không bị ảnh hưởng bởi state, nó sẽ **không bị re-render** khi cha cập nhật state.

🔁 **So sánh:**

❌ **Không tối ưu:**

tsx

```
const Counter = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <button onClick={() => setCount(c => c + 1)}>count: {count}</button>
      <PureComponent />  // bị re-render mỗi lần count thay đổi
    </div>
  );
};
```

✅ **Tối ưu bằng children:**

tsx

```tsx
const App = () => (
  <Counter>
    <PureComponent />
  </Counter>
);

const Counter = ({ children }) => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <button onClick={() => setCount(c => c + 1)}>count: {count}</button>
      {children}  // children không re-render lại
    </div>
  );
};
```

🖼️ 4. Image Optimization – Tối ưu ảnh

📌 Gợi ý:

- **Lazy load** ảnh chưa hiển thị bằng loading="lazy":

html

```html
<img src="..." loading="lazy" />
```

- Dùng định dạng **WEBP** cho ảnh nhẹ hơn PNG/JPEG.
- Sử dụng srcset để cung cấp ảnh phù hợp với độ phân giải thiết bị:

html

```html
<img src="img-400.webp" srcSet="img-800.webp 800w, img-400.webp 400w"
sizes="(max-width: 600px) 400px, 800px" />
```

📊 5. Web Vitals – Tối ưu chỉ số hiệu suất của Google

Google dùng **Web Vitals** (LCP, FID, CLS) để xếp hạng website. Kiểm tra qua:

- Lighthouse
- PageSpeed Insights

👉 Theo dõi thường xuyên để cải thiện thứ hạng SEO và UX.

🗂️ 6. Data Prefetching – Dự đoán và tải dữ liệu trước

📌 Mục tiêu:

- Nếu biết người dùng sắp truy cập một trang → tải trước dữ liệu → khi vào trang, dữ liệu đã sẵn sàng.

🛠️ Với React Query:

ts

```
// Trong sự kiện hover hoặc pre-navigation:
queryClient.prefetchQuery(['product', id], () => fetchProduct(id));
```

➡️ Cực kỳ hữu ích cho menu, link hover, breadcrumbs,…

🧠 Tổng kết

Tối ưu	Kỹ thuật chính
Tải nhanh hơn	Code splitting theo routes, lazy loading
Giảm re-render	State gần component, children prop, memo
Tối ưu style	Dùng Tailwind/CSS module thay vì styled-components
Tối ưu ảnh	lazy load, webp, srcset
Prefetch dữ liệu	queryClient.prefetchQuery()
Theo dõi hiệu năng	Lighthouse, Web Vitals, Chrome DevTools