Báo cáo cải tiến

Cải thiển giao diện và chức năng.

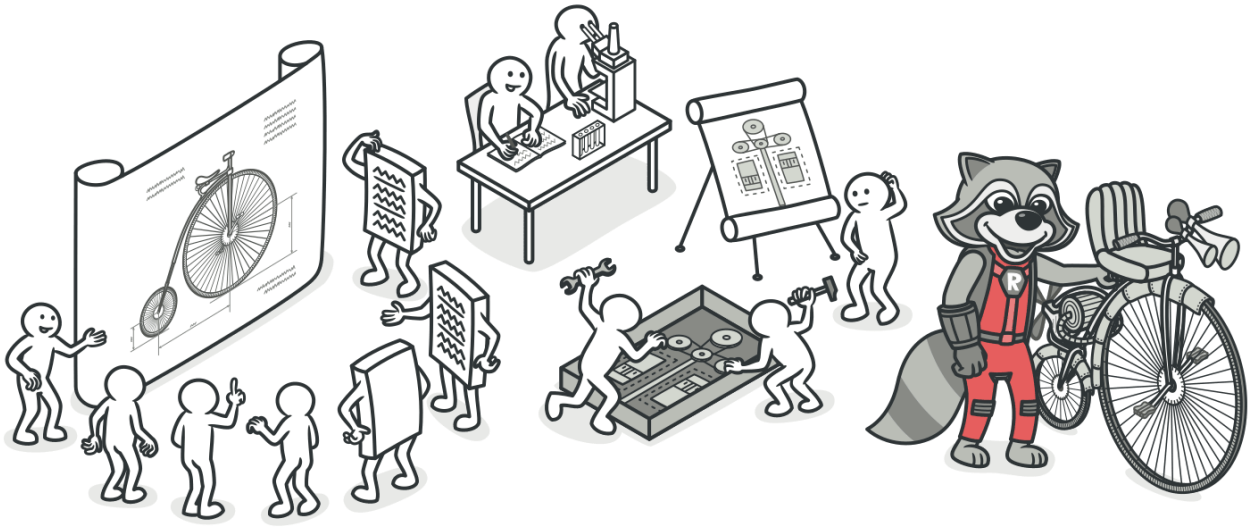
Giao diện ban đầu: đơn giản, khó thao tác, cần phải xác định rõ điểm đến.

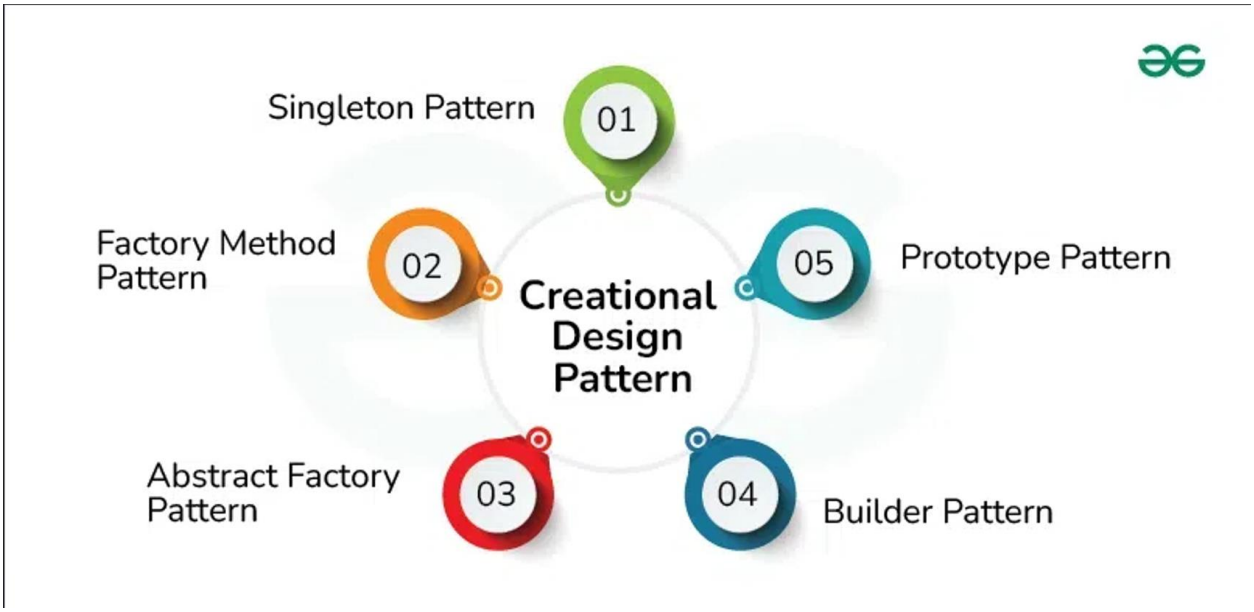
Ảnh đầu: login, signup, find room

Sau: sử dụng Reactjs, Vuejs, Tailwind CSS)

Kết quả: UI Dễ nhìn, tăng UX và tốc độ tải trang giúp giảm thời gian tải lại trang khi chuyển đổi giữa các chức năng. Tối ưu hóa bố cục bảng, form nhập liệu, và tìm kiếm bằng **Axios** và **React hooks** giúp giao diện mượt mà hơn.

Kết quả: Giao diện trực quan hơn, cải thiện trải nghiệm người dùng với thời gian phản hồi nhanh và tương thích trên mọi thiết bị.





Design Pattern là một giải pháp tổng thể cho các vấn đề chung trong thiết kế phần mềm. Nó cũng tương tự các bản thiết kế cho xây dựng nhà cửa, chúng được dùng để giải quyết các vấn đề lặp đi lặp lại trong thiết kế của bạn.

Các design pattern không thể copy rồi paste như cách bạn làm với các function có sẵn hay thư viện, vì chúng không phải là những đoạn code cụ thể. Design pattern ở đây là những khái niệm tổng quát để giải quyết các vấn đề riêng biệt. Bạn có thể tìm hiểu các design pattern và triển khai chúng lên ứng dụng của bạn. Các pattern thường bị nhầm lẫn với thuật toán, vì chúng đều là những khái niệm mô tả giải pháp cho một vấn đề nào đó.

Trong khi thuật toán là định nghĩa những hành động cụ thể để giải quyết vấn đề thì design pattern lại là một mô tả cao hơn cho các giải pháp. Code cho cùng một pattern có thể được triển khai trên hai ứng dụng khác nhau.

Tài liệu design pattern bao gồm những gì ?

Hầu hết các tài liệu mô tả rất chính thống, để cho mọi người có thể tái sử dụng cho nhiều trường hợp. Dưới đây là các thành phần thường có trong các document mô tả pattern:

- **Invention**: mục đích pattern, mô tả ngắn gọn cả vấn đề và giải pháp.
- **Motivation**: giải thích thêm vấn đề và giải pháp mà mô hình khả thi.
- **Structure**: cấu trúc của các lớp cho thấy từng phần của pattern và chúng có liên quan như thế nào.
- **Example**: ví dụ bằng một trong những ngôn ngữ lập trình phổ biến giúp bạn dễ dàng nắm bắt ý tưởng đằng sau pattern.

Tại sao nên học design pattern

Sự thật là các lập trình viên có thể xoay xở làm việc trong nhiều năm mà không cần biết đến bất kỳ pattern nào. Rất nhiều người làm như vậy. Tuy nhiên, ngay cả trong trường hợp đó, bạn có thể đang triển khai một số pattern mà không hề hay biết. Vậy tại sao bạn lại dành thời gian tìm hiểu chúng?

- Giúp sản phẩm của chúng ta linh hoạt, dễ dàng thay đổi và bảo trì hơn.
- Có một điều luôn xảy ra trong phát triển phần mềm, đó là sự thay đổi về yêu cầu. Lúc này hệ thống phình to, các tính năng mới được thêm vào trong khi performance cần được tối ưu hơn.
- Design pattern cung cấp những giải pháp đã được tối ưu hóa, đã được kiểm chứng để giải quyết các vấn đề trong software engineering. Các giải pháp ở dạng tổng quát, giúp tăng tốc độ phát triển phần mềm bằng cách đưa ra các mô hình test, mô hình phát triển đã qua kiểm nghiệm.
- Những khi bạn gặp bất kỳ khó khăn đối với những vấn đề đã được giải quyết rồi, design patterns là hướng đi giúp bạn giải quyết vấn đề thay vì tự tìm kiếm giải pháp tốn kém thời gian.
- Giúp cho các lập trình viên có thể hiểu code của người khác một cách nhanh chóng (có thể hiểu là các mối quan hệ giữa các module chẳng hạn). Mọi thành viên trong team có thể dễ dàng trao đổi với nhau để cùng xây dựng dự án mà không tốn nhiều thời gian.

Khi nào nên sử dụng design pattern

Việc sử dụng các design pattern sẽ giúp chúng ta giảm được thời gian và công sức suy nghĩ ra các cách giải quyết cho những vấn đề đã có lời giải. Lợi ích của việc sử dụng các mô hình Design Pattern vào phần mềm đó chính là giúp chương trình chạy uyển chuyển hơn, dễ dàng quản lý tiến trình hoạt động, dễ nâng cấp bảo trì, …

Tuy nhiên điểm bất cập của design pattern là nó luôn là một lĩnh vực khá khó nhằn và hơi trừu tượng. Khi bạn viết code mới từ đầu, khá dễ dàng để nhận ra sự cần thiết phải có design pattern. Tuy nhiên, việc áp dụng design pattern cho code cũ thì khó khăn hơn.

Khi sử dụng những design pattern có sẵn thì chúng ta sẽ đối mặt với một vấn đề nữa là perfomance của product (code sẽ chạy chậm chẳng hạn). Cần phải chắc chắn là bạn đã hiểu toàn bộ mã nguồn làm việc như thế nào trước khi đụng vào nó. Việc này có thể là dễ dàng hoặc là đau đầu, phụ thuộc vào độ phức tạp của code.

Hiện nay chúng ta đang áp dụng rất nhiều design pattern vào công việc lập trình của mình. Nếu bạn thường tải và cài đặt các thư viện, packages hoặc module nào đó thì đó là lúc bạn thực thi một design pattern vào hệ thống.

Tất cả các framework cho ứng dụng web như Laravel, Codeigniter… đều có sử dụng những kiến trúc design pattern có sẵn và mỗi framework sẽ có những kiểu design pattern riêng.

Lý do sử dụng Design Pattern là gì?

Sau khi biết về định nghĩa Design Pattern là gì, hiểu được lý do sử dụng Design Pattern la gi cũng vô cùng cần thiết. Một số chia sẻ sau đây của chúng tôi có thể làm cho bạn đọc hiểu được về vấn đề này.

- Phần mềm giúp cho các lập trình viên có thể hiểu sâu thêm về các mã code của người khác một cách nhanh chóng hơn nhiều lần. Có thể nói cách khác là hiểu về những mối quan hệ giữa các Module với nhau. Các thành viên ở trong team sẽ trao đổi với nhau một cách dễ dàng và cùng nhau xây dựng các dự án mà không mất quá nhiều thời gian.
- Nếu bạn gặp bất cứ khó khăn gì với vấn đề sau khi mình đã giải quyết, sử dụng các loại Patterm này sẽ là cách hữu hiệu giúp bạn giải quyết các khúc mắc thay vì phải tìm những giải pháp tốn kém, mất nhiều thời gian.
- Các giải pháp do Design Pattern mang đến đều đã được tối ưu hóa một cách tối đa, hơn nữa kiểm chứng rõ ràng vài giải quyết tốt trong software engineering. Với dạng tổng quát này, phần mềm có thể tăng tốc độ phát triển bằng cách đưa ra nhiều mô hình khác nhau, bao gồm mô hình phát triển, mô hình test khi đã qua kiểm nghiệm.
- Nhờ có Design Pattern mà sản phẩm được linh hoạt hơn qua nhiều khâu, có thể dễ dàng mang đi bảo trì và thay đổi nếu có rủi ro xảy ra
- Việc phát triển phần mềm luôn gắn liền với những yêu cầu thay đổi. Hệ thống khi phình to ra sẽ có những tính năng mới được thêm vào.

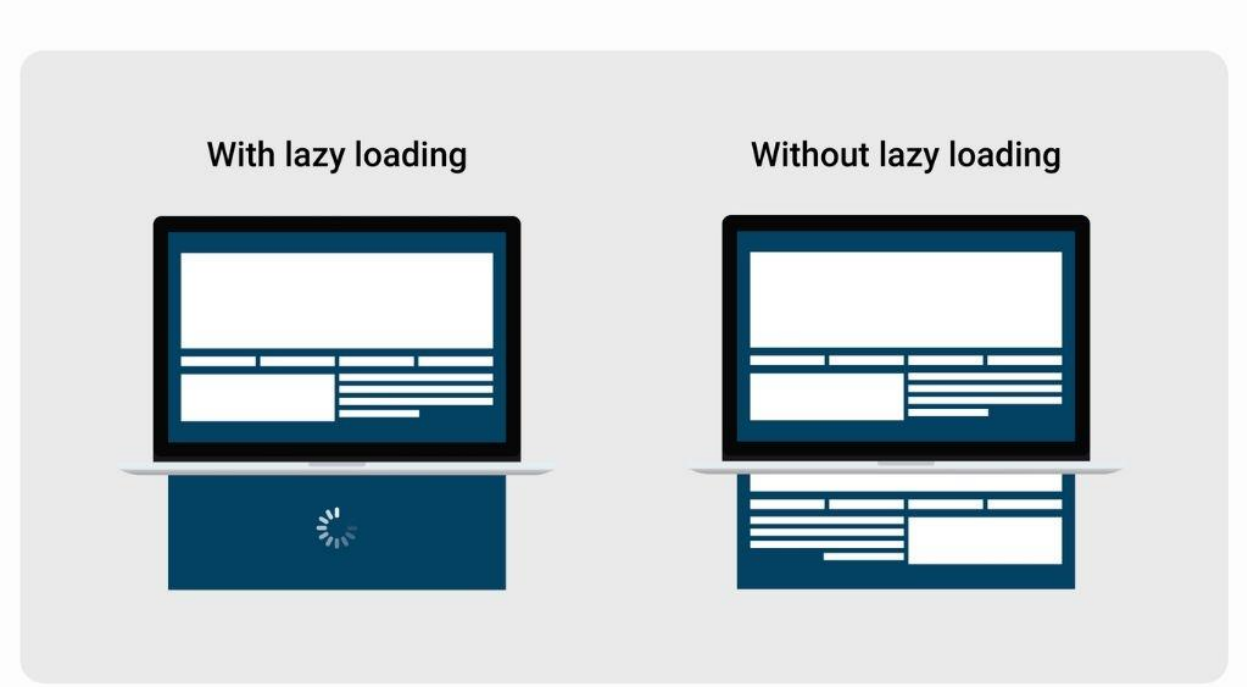
Lazy loading là gì?

Lazy loading là động tác khởi lệnh của trình duyệt. Nếu trang có nhiều dữ liệu cần tải, để load toàn bộ cùng một lúc thì tốc độ trang sẽ chậm, ảnh hưởng đến khả năng truy cập của người dùng và chất lượng trang.

Hầu hết người truy cập không kéo xuống cuối trang để đọc nội dung, nên việc load toàn bộ dữ liệu sẽ làm tốn nguyên liệu của website.

Ưu điểm:

Lazy loading là một trong những kỹ thuật tối ưu hóa hiệu suất trang web được sử dụng phổ biến, hạn chế sự chậm trễ khi tải đồng loạt dữ liệu trên trang. Vì tính năng này chỉ tải những dữ liệu cần thiết trong tầm nhìn của người dùng.



Tại sao cần dùng lazy loading?

Nâng cao website performance

Nếu tất cả dữ liệu được load cùng một lúc khi truy cập trang thì trình duyệt sẽ mất rất nhiều thời gian để tải. Tệ hơn nữa, trang bị lag và phải đợi trình duyệt tải lại từ đầu.

Ngoài ra, nếu trình duyệt bị quá tải sẽ phát sinh lỗi và không thể tải một số dữ liệu, dẫn đến dữ liệu bị ẩn trên web. Điều này tạo ra ấn tượng xấu về hình thức và khiến người dùng cảm thấy trang web kém hấp dẫn. Dẫn đến lưu lượng truy cập trang web và tương tác giảm. Lazy loading chỉ tải dữ liệu khi người dùng lướt tới đâu thì nội dung hiện tới đó, cải thiện trải nghiệm của người dùng và trang web "bảo toàn" tất cả dữ liệu.

Tiết kiệm tài nguyên

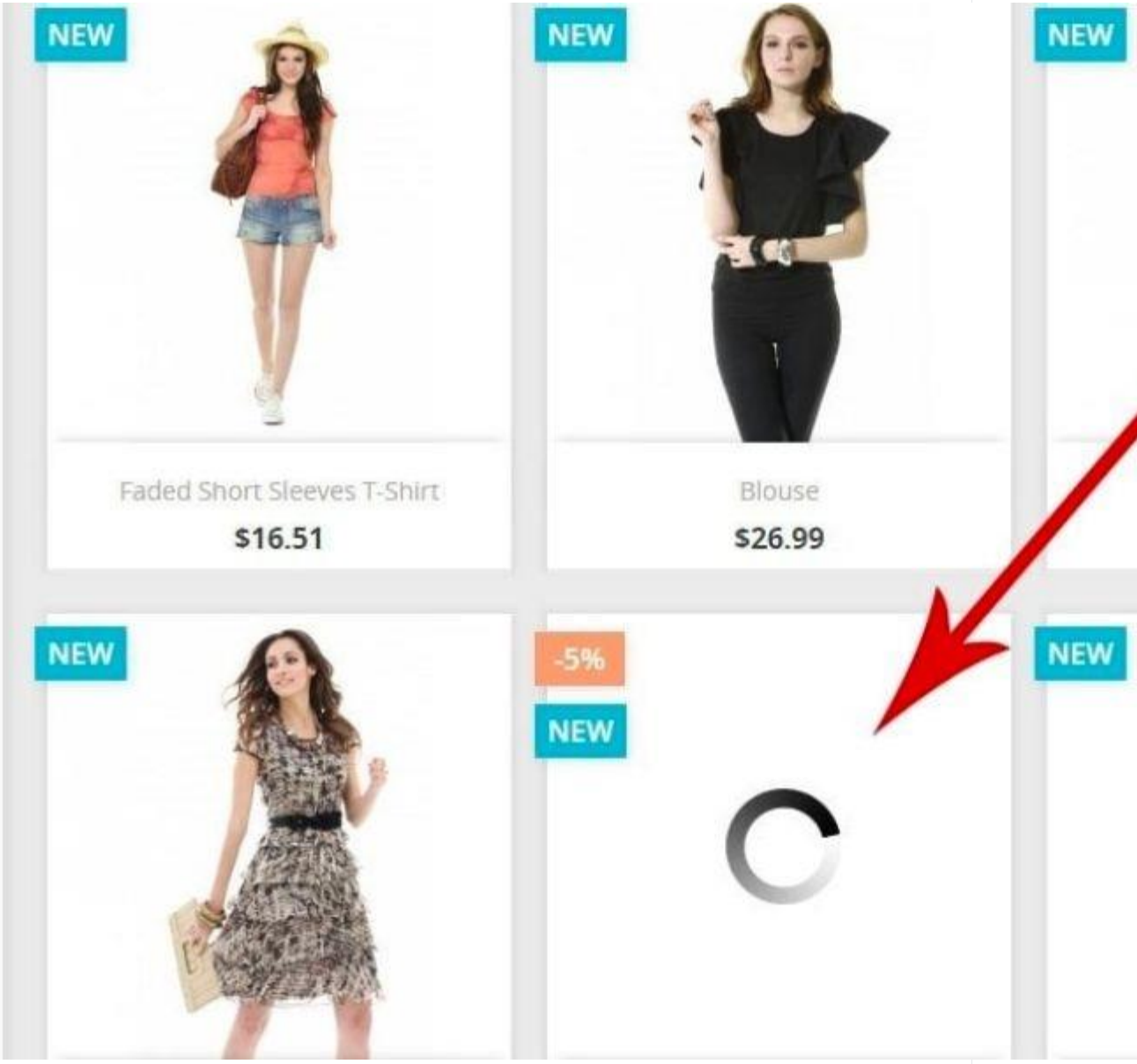
Sử dụng lazy loading cho dữ liệu chưa cần sẽ tiết kiệm bộ nhớ, CPU, GPU,... và hữu ích cho người dùng sử dụng trình duyệt trên di động có kết nối chậm.

Cải thiện trải nghiệm người dùng

Trải nghiệm người dùng (UX – User Experience) có nhiều tiêu chí để xác định một trang web có đáp ứng được trải nghiệm người dùng hay không.

Một trong những tiêu chí đó là tốc độ tải trang. Khi một website phải tải những dữ liệu không cần thiết sẽ làm chậm tốc độ tải của website và dữ liệu trên trang.

Ngoài ra, số lượng người dùng truy cập internet qua thiết bị di động ngày càng tăng. Vì vậy, tối ưu hóa thiết bị di động cũng là một trong những tiêu chí cần được chú ý. Lazy loading giúp tăng tốc độ tải trang, giảm thiểu lỗi dữ liệu ẩn và cải thiện trải nghiệm người dùng một cách hiệu quả.



Gia tăng điểm đánh giá website

Người dùng và cả Google đều ưu tiên các trang web tải nhanh hơn những trang web chậm vì cung cấp kết quả tìm kiếm nhanh hơn, từ đó đạt nhiều lượt truy cập hơn.

Nếu bạn quan tâm đến điểm đánh giá trang web, lazy loading cũng là kỹ thuật cải thiện điểm số trang web hiệu quả.

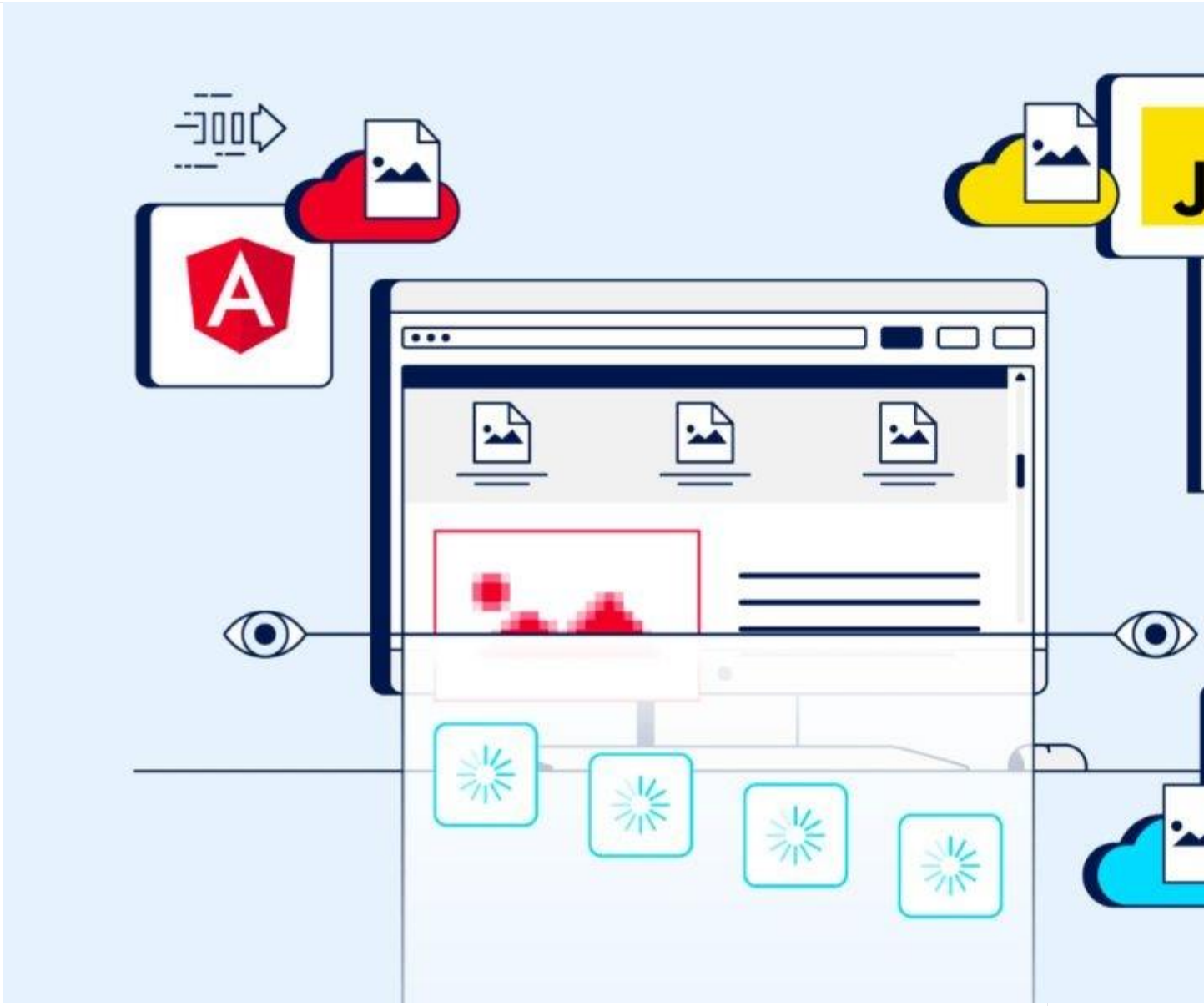
Lazy loading giúp trang web tải nhanh hơn vì không cần load tất cả dữ liệu. Điều này sẽ làm cho trang web được đánh giá cao hơn bởi thang đo tốc độ tải trang.

Xem thêm: *Framework là gì? Tuyển tập những điều bạn cần biết về nền tảng lập trình Framework*

Cách sử dụng lazy loading

Lazy loading chỉ thực sự phát huy tác dụng khi được sử dụng đúng mục đích, đúng thời điểm và đúng đối tượng. Bạn nên chú ý những điều sau:

- Các nội dung chưa được tải, có hiện tượng nhấp nháy.
- Quá trình lazy-load yêu cầu nhiều Javascript hơn, phức tạp hơn và dễ xảy ra lỗi hơn. Nếu Javascript không được tải xuống hoặc không thể chạy do lỗi kết nối mạng thì dữ liệu được khởi lệnh lazy-load sẽ không hiển thị.



Trường hợp nên dùng lazy loading

Nếu trang web có quá nhiều dữ liệu để tải sẽ khiến trang bị chậm, giảm hiệu suất trang web.

Trang web nên được tối ưu hóa cho điện thoại để cải thiện tốc độ tải trang. Người dùng sử dụng các thiết bị được kết nối băng thông rộng, tốc độ kết nối mạnh để tránh lỗi Javascript của lazy-load.

Nếu trang web có dữ liệu bán hàng (một trang web tương tự như trang thương mại điện tử), việc áp dụng lazy-load sẽ không phù hợp vì khách hàng có thể không tìm thấy sản phẩm cần nhưng bị ẩn.

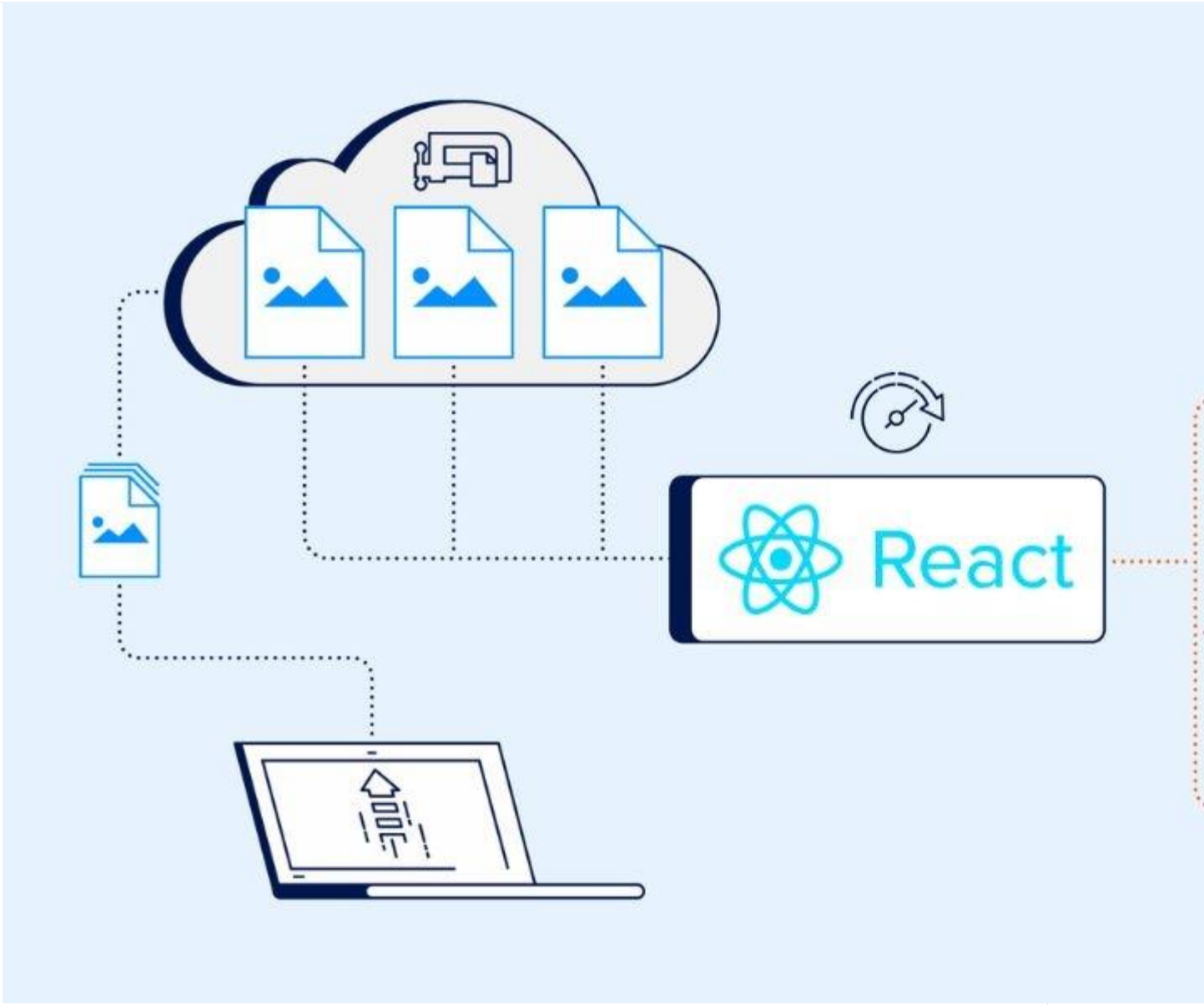
Chuẩn bị dùng lazy loading

Trước khi sử dụng bất kỳ kỹ thuật nào, bạn phải hiểu kỹ thuật đó có tính năng và đem lại lợi ích gì cho trang web của bạn, tương tự lazy loading cũng vậy.

Lazy loading đơn giản là một thuộc tính để điều khiển sự xuất hiện của các dữ liệu trên trang web như hình ảnh. Bạn có thể sử dụng lazy-load với thuộc tính loading.

Thuộc tính loading có 3 giá trị tương ứng với chức năng của nó.

- **Lazy - lazy-loading:** Trình duyệt cần lazy-load dữ liệu này.
- **Eager - eager:** Giá trị này yêu cầu trình duyệt tải dữ liệu ngay lập tức, càng nhanh càng tốt. Nếu dữ liệu đang ở chế độ lazy được chuyển sang eager sẽ lập tức được tải ngay.
- **Auto – auto:** Trình duyệt quyết định có nên lazy-load dữ liệu hay không.



Dùng thuộc tính loading

Cách áp dụng tính năng này khá đơn giản mà không cần dùng Javascript và lo lắng về lỗi hình ảnh trên trang. Điều cần làm là báo hiệu cho trình duyệt những hình ảnh nào cần lazy-load bằng thuộc tính loading hoặc 3 giá trị tương ứng lazy, eager và auto.

Dùng Fallback hoặc Polyfill

Với browser không hỗ trợ loading, cách sử dụng lazy-load là dùng Fallback (dự phòng) và Polyfill (một mã triển khai tính năng mà trình duyệt web không hỗ trợ).

Với Fallback hay Polyfill, bạn viết code với thuộc tính loading để thực hiện lazy-load hình ảnh.

Dùng Intersection Observer API

Cách dùng Intersection Observer API như sau:

- API Intersection Observer cho phép bạn giám sát dữ liệu (hình ảnh) khi người dùng lướt tới.
- Intersection Observer API hỗ trợ lazy-load hình ảnh nhưng không có thuộc tính loading. Chỉ sử dụng lệnh lazy image các hình ảnh trên trang. Ngoài ra, API Intersection Observer cũng lazy-load cả ảnh background.

Những lưu ý khi sử dụng lazy loading

Với mỗi tính năng khác nhau sẽ có đặc điểm riêng và phù hợp với các trình duyệt web khác nhau. Ví dụ: API Intersection Observer không hỗ trợ Internet Explorer và Opera Mini mặc dù đều hỗ trợ lazy-load cho các trình duyệt web khác.

Đối với Internet Explorer, để sử dụng lazy-load các dữ liệu bạn phải dùng Polyfill để giả lập API Intersection Observer và sử dụng như bình thường.

Về layout shift:

- Nếu sử dụng lazy loading cho hình ảnh và dữ liệu nói chung, khi các dữ liệu được tải cùng lúc xem của người dùng thì có thể xuất hiện trường hợp nhảy thông tin vì trình duyệt không thể tính toán kích thước của hình ảnh. Kết quả là các thông tin được đặt lung tung gây mất thẩm mỹ.
- Do đó, trước khi dùng lazy loading bạn cần chỉ ra chính xác kích thước hình ảnh sẽ hiển thị khi người dùng lướt đến. Điều này giúp hình ảnh được tải mà không làm xê dịch thông tin trên trang.

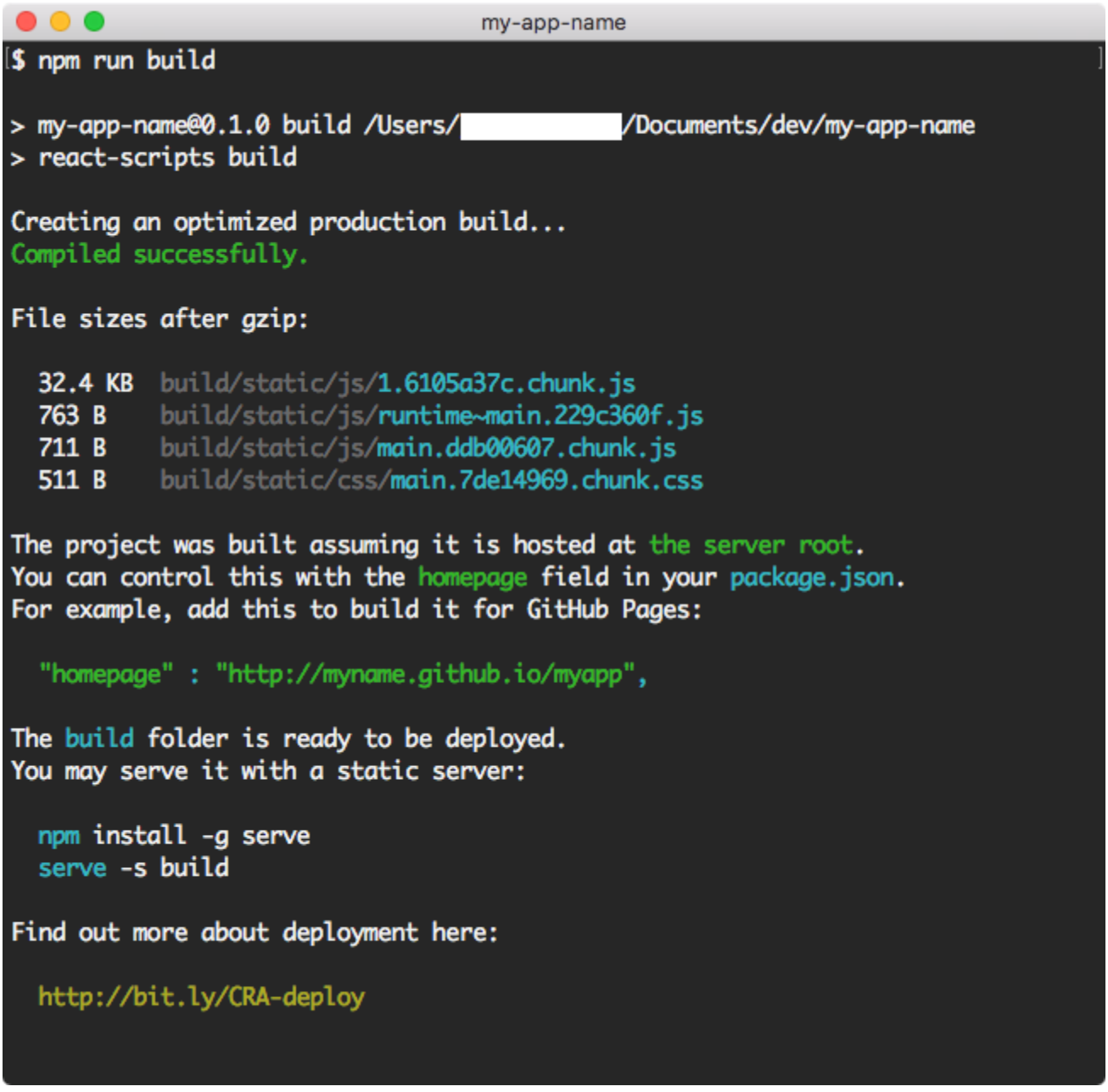
Không nên lạm dụng lazy-loading. Nếu trang web có ít hình ảnh (dưới 5) thì không nên sử dụng tính năng này vì không ảnh hưởng đáng kể đến tốc độ tải trang. Bạn có thể tìm cách tối ưu các hình ảnh thay vì lazy loading và cẩn thận khi sử dụng API Intersection Observer để thực hiện lazy-load

Khi dùng API Intersection Observer để lazy-load, lệnh scr sẽ trở thành data-scr và Googlebot không hiểu data-scr là gì. Do đó, Googlebot coi hình ảnh này bị lỗi và không index hình ảnh đó. Đối với các giá trị lazy của thuộc tính loading, cách này không áp dụng lazy-load cho background. Nên tránh lazy-load các hình ảnh đầu trang để giảm thiểu layout shift.

Bundle JS Files là gì?

Trước tiên bạn cần hiểu khái niệm bundle file trong React là gì. Thông thường khi tạo ra một ứng dụng thì chúng ta sẽ viết source code của mình vào nhiều files khác nhau, trong đó có chứa nhiều các modules và thư viện bên thứ 3 (3rd-party

libs). Khi tiến hành build ứng dụng của bạn, React sẽ thực hiện việc chuyển đổi rất nhiều file source code mà bạn viết trở thành 1 file lớn hơn để sử dụng nó đưa cho các trình duyệt web khi load ứng dụng. Những file đó được gọi là bundle.



Nguồn: telerik.com

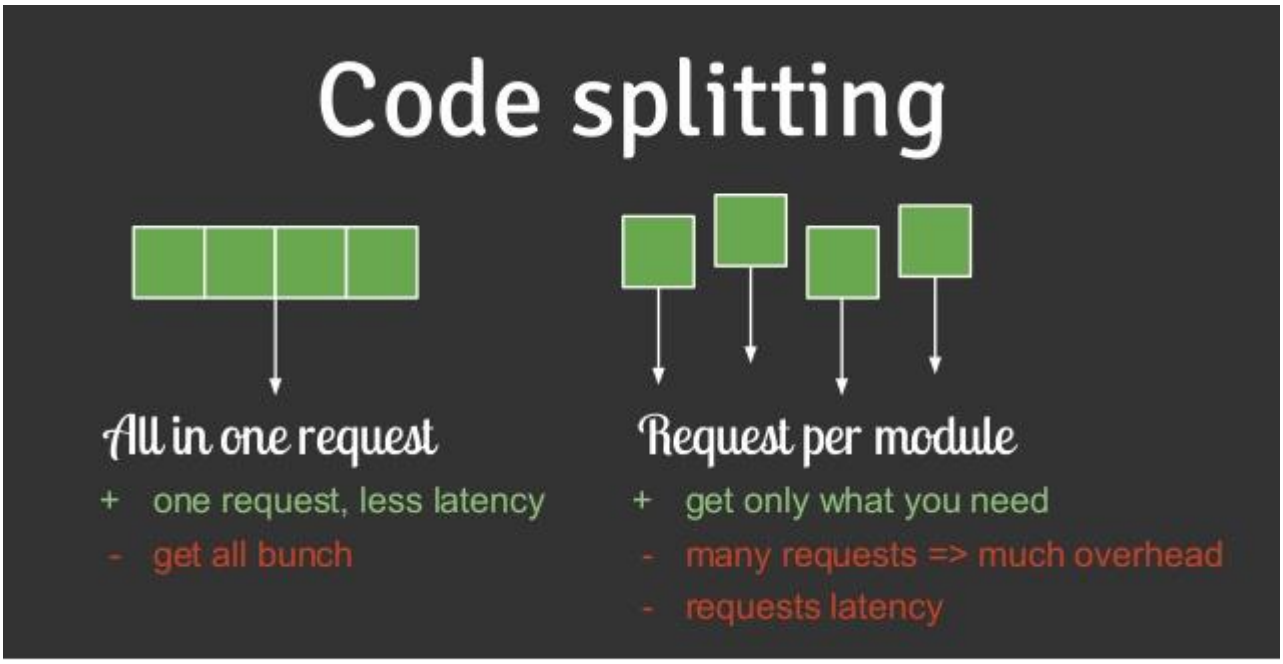
Ban đầu ứng dụng của bạn tạo ra những file bundle nhỏ, và việc trình duyệt load chúng lên không thành vấn đề; sau một thời gian phát triển, với việc import và sử dụng ngày càng nhiều các thư viện và module thì các files bundle của bạn cũng

ngày càng nặng thêm. Nếu không thực hiện tối ưu, kích thước các file bundle có thể lên tới 40-50Mb là chuyện bình thường, điều đó cũng đồng nghĩa với việc ứng dụng của bạn trở nên nặng nề khi load, user sẽ cần chờ 1 khoảng thời gian khá lâu để có thể tương tác được với các phần tử trên màn hình.

Vậy cách giải quyết cho vấn đề này là gì?

Code-Splitting là gì?

Rõ ràng để load một ứng dụng hay một màn hình cụ thể, ứng dụng của bạn không cần phải "nạp" hết các module hay thư viện được import vào; vì thế để giải quyết cho vấn đề bundle file size lớn, chúng ta cần một kỹ thuật để tách nó ra thành 2 phần: phần cần thiết load để khởi động ứng dụng (hay màn hình) và phần có thể load nạp vào sau khi ứng dụng đã được chạy. Và khi cần cần thiết load để có thể khởi động ứng dụng càng nhỏ, thì thời gian tải ứng dụng của chúng ta càng nhanh hơn. React đã cung cấp cho chúng ta tính năng này và gọi nó là Code-Splitting.



Nguồn: wiki.tino.org

Có 3 kỹ thuật xử lý trong Code-Splitting thường được sử dụng, chúng ta cùng lần lượt tìm hiểu và xem cách triển khai của chúng nhé.

Dynamic Import

Thông thường khi chúng ta cần import 1 module nào để sử dụng, câu lệnh import sẽ được thực hiện như dưới đây:

```
import { add } from './math';




console.log(add(16, 26));
```

Đoạn khai báo trên sẽ import module math một cách đồng bộ – tức là sẽ import vào luôn file bundle từ khởi tạo. Với Dynamic Import, chúng ta sẽ xử lý lại đoạn code trên để chỉ import khi ứng dụng cần gọi đến phương thức add của module math. Code sẽ được viết lại như sau:

```
import("./math").then(math => {

  console.log(math.add(16, 26));

});
```

Phương thức dynamic import giúp việc import module, file một cách bất đồng bộ bằng việc trả về 1 Promise. Nó hoạt động được cả ở server-side và client-side giúp bạn có thể sử dụng trong các trường hợp load file, assets hay những module, 3rd-party libs không cần thiết cho việc hiển thị ứng dụng, màn hình lần đầu tiên.

Sử dụng React.lazy và Suspense

Phương thức React.lazy giúp bạn tạo ra 1 component ở dạng lazy-loading, nghĩa là sẽ chỉ tạo ra component đó khi nó thực sự được gọi đến và cần hiển thị ra. Hãy xem ví dụ dưới đây:

```
import React, { useState } from 'react';
import ProjectIntro from './projectIntro';
import ProjectDetails from './projectDetails';

export default function App() {
  const [showDetails, setShowDetails] = useState(false);

  return (
      <>
        <h1>Project List</h1>
        <ProjectIntro />
```

```
        <button     onClick={()     =>     setShowDetails(true)}>Show
Details</button>
        {showDetails ? <ProjectDetails /> : null }
      </>
  );
};
```

Xem tiếp...

Ở đoạn code trên, component ProjectDetails mặc định sẽ không hiển thị, tuy nhiên nó vẫn sẽ được load vào trong bundle file vì đã được import ngay trên đầu. React.lazy giúp bạn dynamic import 1 component, kết hợp với Suspense bọc bên ngoài cho phép chúng ta thêm xử lý hiệu loading component đó mà không làm tăng đáng kể bundle file size. Source code cho việc triển khai React.lazy và Suspense như sau:

```
import React, { useState, Suspense } from 'react';
import ProjectIntro from './projectIntro';

const ProjectDetails = React.lazy(() => import("./projectDetails"));

export default function App() {
  const [showDetails, setShowDetails] = useState(false);

  return (
    <>
        <h1>Project List</h1>
        <ProjectIntro />
        <button     onClick={()     =>     setShowDetails(true)}>Show
Details</button>
        <Suspense fallback={<div>Loading...</div>}>
          {showDetails ? <ProjectDetails /> : null }
        </Suspense>
    </>
  );
};
```

Xem tiếp...

Có một lưu ý ở đây là React.lazy sẽ không thực hiện được khi render app ở server-side. Các bạn có thể tham khảo cách sử dụng 1 thư viện lazy load khác dành cho React như Loadable-components.

Route-based code splitting

Ở hai kỹ thuật trên, chúng ta đã tìm cách giảm kích thước bundle file bằng cách giảm những thành phần cần thiết để load ứng dụng trong 1 component. Với phạm

vi một ứng dụng, chúng ta có chứa nhiều màn hình, tại mỗi thời điểm sử dụng thì người dùng sẽ chỉ tương tác với một hoặc một vài màn hình nhất định. Vì thế việc code splitting hoàn toàn cần thiết để thực hiện trên phạm vi ứng dụng.

Khi 1 ứng dụng React chạy, tương ứng với mỗi route sẽ có 1 component đảm nhiệm việc hiển thị và tương tác với người dùng, điều đó cũng có nghĩa các component khác không cần thiết phải được load lên ngay lúc đó. Chúng ta vẫn sẽ sử dụng lazy load cho trường hợp này, code thực hiện sẽ như sau:

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

const Profile = lazy(() => import('./profile'));
const ContactUs = lazy(() => import('./contact'));

const App = () => (
  <Router>
    <Suspense fallback={<div>Loading...</div>}>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/contact" element={<ContactUs />} />
      </Routes>
    </Suspense>
  </Router>
);
```

Xem tiếp...

Ở đoạn code trên, 2 components Profile và ContactUs được xử lý lazy loading, chúng sẽ chỉ được load khi user đi đến route tương ứng. Điều đó giúp cho ứng dụng của bạn không cần phải chờ load hết tất cả các component trong Router, giảm kích thước bundle file ban đầu.

Lời kết

Như vậy chúng ta đã đi qua được các kỹ thuật **Code-Splitting giúp tối ưu ứng dụng React** thông qua việc giảm bundle file size. Trong thực tế các project React hiện nay, việc sử dụng lazy load là hết sức cần thiết khi có quá nhiều component, các module, thư viện được sử dụng trên cùng 1 màn hình. Hãy cố gắng tối ưu source code của bạn nhất có thể ngay từ ban đầu để tránh phải giải quyết các vấn đề về hiệu năng cho việc mở rộng sau này. Hy vọng bài viết cung cấp cho bạn kiến thức hữu ích cho các dự án React sắp tới, hẹn gặp lại mọi người trong các biết tiếp theo của mình.