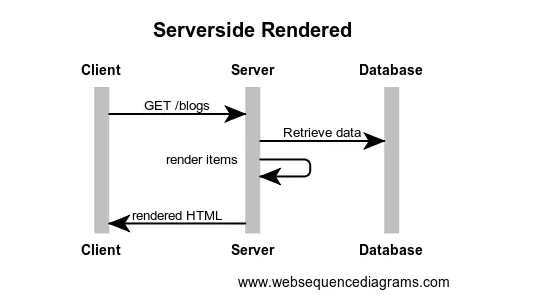
**BÁO CÁO 3: SEVER SIDE RENDERING & CLIENT SIDE RENDERING, NEXTJS, GASBY VÀ GENERATOR FUNCTION**

1. **Server side rendering & client side rendering**
2. **Server side rendering**

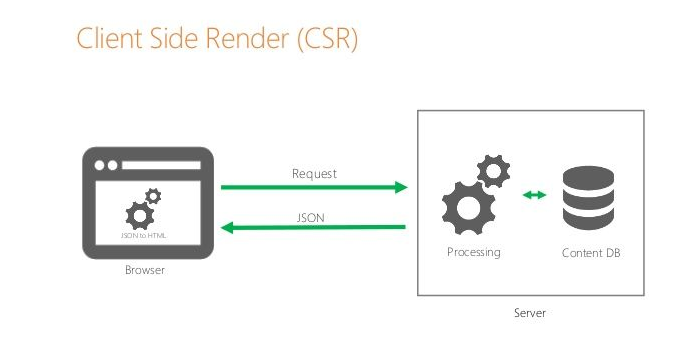
* Đây là cơ chế được sử dụng từ lâu. Các xử lý về logic sẽ được thực hiện ở phía server và client chỉ nhận các kết quả trả về từ phía server này.
* Cụ thể hơn như sau:
  + Khi người dùng truy cập vào một trang web. Ngay lập tức trình duyệt sẽ gửi GET request đến server.
  + Server sau khi nhận request sẽ xử lý các yêu cầu và lấy dữ liệu từ database để xử lý.
  + Cuối cùng server sẽ render HTML rồi trả về cho trình duyệt để hiển thị.



* Ưu điểm của server side rendering:
  + Việc khởi tạo các thành phần khá nhanh và dễ optimize vì toàn bộ dữ liệu đã được xử lý ở server. Client chỉ làm nhiệm vụ hiển thị.
  + Đa số cái web framework đều hỗ trợ cơ chế này
  + Dễ hiểu và dễ code hơn do không cần phải chia ra frontend và backend
  + Chạy được trên phần lớn các trình duyệt. Kể cả disable JS vẫn chạy được.
* Nhược điểm của server side rendering:
  + Do mỗi lần chuyển trang thì client lại phải gửi request cho server trả về HTML nên phải mất rất nhiều thời gian tải lại mỗi lần chuyển trang. Do đó, nó cũng gây khó chịu cho người dùng.
  + Nặng cho phía server vì phải gánh nhiều thành phần xử lý logic và dữ liệu.
  + Tốn băng thông do server gửi nhiều thành phần trùng lặp (khung trang, menu, header, footer, …).

1. **Client side rendering**

* Đây là cơ chế được sử dụng trong Single Page Application (SPA). Ngược lại với server side rendering, client side rendering sẽ xử lý một số thành phần đơn giản (render html, css, js)
* Cụ thể hơn như sau:
  + Những logic đơn giản (validation, đọc dữ liệu, sorting, filtering) nằm ở phía client
  + Logic để chuyển trang (routing), render dữ liệu sẽ nằm ở phía client, trừ một số routing đặc biệt do server quản lý
  + Một số logic phức tạp (thanh toán, phân quyền) vẫn được xử lý ở phía server



* Ưu điểm của client side rendering
  + Page chỉ cần load 1 lần duy nhất. Khi user chuyển sang trang khác hoặc thêm dữ liệu, client sẽ gửi request và lấy dữ liệu từ phía server qua AJAX. Từ đó, client có dữ liệu mới mà không cần chuyển trang
  + Do chuyển một số logic cho client nên phía server sẽ đỡ nặng hơn
  + Giảm được băng thông do client chỉ cần một phần dữ liệu, không cần toàn bộ dữ liệu của cả trang
  + Với ứng dụng cần nhiều tương tác, single page application hoạt động mượt mà hơn trên browser mà không cần load đi load lại nhiều
* Nhược điểm:
  + Việc khởi tạo trang mất nhiều thời gian hơn vì browser phải tải toàn bộ JS về, gọi API để lấy dữ liệu về server rồi mới render dữ liệu
  + Đòi hỏi cấu trúc của phần mềm phải có 2 phần riêng biệt là Front end và Back end nên việc code sẽ phức tạp hơn
  + Không chạy được nếu như JS bị disable hoặc ở các trình duyệt cũ không nhận được JS ES6
  + Nếu phía client sử dụng mobile có cấu hình yếu thì sẽ bị chậm khi load.

1. **Nextjs**
2. **Dẫn nhập**

* Cơ chế Client side rendering được sử dụng chính trong React, chính vì thế với mỗi trang đơn giản (chẳng hạn như About) thì client phải tải một lượng lớn JS về. Chính vì sự bất tiện này Server side rendering dành cho React ra đời và điều này làm tăng hiệu suất hơn cho các điện thoại với cấu hình yếu

1. **Giới thiệu**

* Trong Next.js có những giải pháp sau:
  + Tốt cho việc SEO website
  + Tích hợp bộ nhớ đệm và tối ưu hóa tĩnh tự động
  + Các trang hoàn toàn do máy chủ hiển thị
  + 100% React hỗ trợ
  + Tinh chỉnh cấu hình webpack / babel của bạn nếu cần

1. **Các tính năng**

* Tạo các nội dung tĩnh bằng cách động: khi Next.js tìm nạp URL cụ thể, nó sẽ lưu nó dưới dạng một trang tĩnh và phân phát nó một cách tĩnh bất cứ khi nào ai đó truy cập đường dẫn. Đồng thời, nó sẽ sẵn sàng chấp nhận các paths mới một cách linh hoạt.
* Webpack 5 support
* Trong React thông thường sẽ có một hàm duy nhất thực thi cả 2 môi trường server và client, đó là getInitialProps. Tuy nhiên Next.js có giải pháp là đưa ra 2 phương thức là getServerSideProps (đưa các props vào trang từ server) và getStaticProps (tạo ra các đầu ra tĩnh tại thời điểm build).
* Hỗ trợ bộ nhớ đệm liên tục cho các trang không bị thay đổi
* Hỗ trợ cho SCSS

1. **Gasby**
2. **Giới thiệu**

* Gastby JS là nền tảng dùng để xây dựng website và web app tĩnh để hoạt động ở hiệu suất rất cao. Với các web động, việc tạo cache ở thời điểm runtime vì thế server phải chạy liên tục để có đầy đủ cơ sở dữ liệu. Còn với Gasby.js thì khi web build xong sẽ ra tập tin tĩnh thi máy chủ có thể tắt đi giúp tiết kiệm chi phí
* Với Gasby.js khi nhấn vào liên kết để chuyển trang thì chỉ cần tải tập tin JSON của nội dung mà trang khác khác cần hiển thị.

1. **Ưu điểm**

* Triển khai với chi phí thấp và dễ dàng
* Tốc độ cao
* Tối ưu SEO
* Hỗ trợ Progressive Web App
* An toàn

1. **Nhược điểm**

* Với việc triển khai hệ thống bình luận, tìm kiếm phải dựa vào các dịch vụ bên thứ 3
* Thời gian build khá lâu
* Triển khai cho người dùng bình thường còn phức tạp. Thật khó để sử dụng chức năng hẹn giờ xuất bản, hoặc chỉ là những cập nhật nho nhỏ.
* Với wordpress, hay 1 nguồn nội dung khác, thì phần “hình ảnh trong bài viết” có thể không tận dụng được khả năng xử lý hình ảnh của Gatsby Js.
* Mất đi một số tính năng hay của các nền tảng khác cung cấp. Ví dụ với WordPress mình kết hợp với OneSignal, khi mỗi bài viết được xuất bản thì sẽ tự động gởi thông báo đến tất cả các người dùng đang theo dõi (subscribers).

1. **Generator function**
2. **Giới thiệu**

* Generator function là một hàm có thể được thực thi nhiều lần liên tiếp mà các số lượng biến, giá trị biến, trạng thái các thành phần bên trong hàm, … đều có thể lưu lại sử dụng sau mỗi phiên. Generator function có thể dừng lại ở bất kì khi nào đợi một điều kiện nào đó xảy ra rồi mới thực thi.

1. **Cú pháp**

function\* name([param[, param[, ...param]]]) {

  //code

}

* Trong đó: name là tên hàm, param : các tham số truyền vào (tối đa 255 tham số)

1. **Giá trị trả về**

* Với generator function, nó không trả về kiểu dữ liệu cơ bản mà thay vào đó, nó trả về một iterator object. Hàm next() của iterator object được sử dụng để truy xuất các node dữ liệu sau mỗi bước resume lại generator function. Khi đó generator function sẽ thực thi hàm cho đến khi gặp từ khóa yield, hoặc return kế tiếp chưa được duyệt ở bước trước.

1. **Yield**

* Yeild là từ khóa dùng để tạm dừng và cũng để tiếp tục việc thực thi bên trong generator function
* Ví dụ:

function\* generatorFunc(index) {

  while (index < 2) {

    yield index++;

  }

}

const iterator = generatorFunc(0);

console.log(iterator.next());

// log output: {value : 0, done : false}

console.log(iterator.next());

// log output: {value : 1, done : false}

console.log(iterator.next());

// log output: {value : underfined, done : true}

1. **Yield \***

* Yield\* là một dạng ủy quyền thực thi. Nó có thể nhúng mã của một generator function ngay sau nó hoặc là ủy quyền trực tiếp cho một iterator object. Ngoài ra, yield\* là một biểu thức chứ không phải một câu lệnh
* Cú pháp:

yield\* [[expression]]

expression ở đây luôn là dạng iterator object nghĩa là nó có thể là mảng, string, hay là generator function.

* Ví dụ:

function\* g2() {

  yield\* [1, 2];

  return 'abc';

}

var rs;

function\* g() {

   rs = yield\* g2();

}

var iterator = g();

console.log(iterator.next());

// {value: 1, done: false}

console.log(iterator.next());

// {value: 2, done: false}

console.log(iterator.next());

// {value: undefined, done: true},

// g2() đã trả về {value: 'abc', done: true}