Tổng hợp full-snack JS

HTML / CSS :

- Phân biệt display flex vs grid và khi nào sử dụng

  + **display flex** là dùng để tạo ra một container linh hoạt (flex container) và điều khiển cách các phần tử con bên trong nó sắp xếp, căn chỉnh và phân chia không gian, thường dùng cho các layout 1 chiều, chỉ có 1 chiều ngang hoặc dọc.

  + **display grid** là cho phép bạn tạo ra một hệ thống lưới (grid system) để sắp xếp các phần tử con trong một cách linh hoạt hơn. ,có thể sử dụng cả 2 chiều ngang và dọc, thường sử dụng cho các layout dạng lưới phức tạp.

- DOM là gì, các cách query element trong DOM

+ DOM là Document Object Model là khi 1 file HTML đc đọc trên trình duyệt nó sẽ sinh ra DOM vs đối tượng lớn nhất là document để cho JS tương tác và thay đổi các thuộc tính và nội dung các đối tượng trong DOM.

```
Document (Đối tượng lớn nhất)
  ├── html
  ├── head
        |    └── title
         |        └── "DOM Example"
  ├── body
        ├── h1
                      |    └── "Hello, World!"
       └── p
                    └── "This is a paragraph."
```

+ Các cách query elements trong DOM :

  - querySelector: `querySelector` sẽ trả về phần tử đầu tiên phù hợp với id,class, hoặc thẻ html mà mình đang tìm.

  - querySelectorAll : Kết quả trả về là một NodeList (tương tự mảng) chứa tất cả các phần tử phù hợp vs id,class, hoặc thẻ html mà mình đang tìm.

- getElementById : `getElementById` được sử dụng để lấy một phần tử duy nhất bằng cách trỏ tới của thuộc tính `id` của nó.Nó chỉ trả về 1 element theo id đó,vì id đặt là duy nhất.

- getElementsByClassName : `getElementsByClassName` trả về một HTMLCollection (tương tự mảng) chứa tất cả các phần tử có class mà được tìm kiếm.

- Phân biệt thẻ sematic và non-sematic

+ Sematic là thẻ ngữ nghĩa vd như là header, main , footer, section v..v… thẻ được thiết kế với mục đích diễn đạt ý nghĩa và cấu trúc của nội dung một cách rõ ràng và có ý nghĩa hơn và nó tốt cho việc SEO trong trang web.

+ Non-sematic là các thẻ không mang theo ý nghĩa rõ ràng hoặc không diễn đạt cấu trúc của nội dung một cách chính xác. Chúng thường được sử dụng để định dạng hoặc trình bày nội dung mà không cung cấp thông tin về ý nghĩa thực sự của nội dung. vd: h1, p ,div …v..v..

- SEO là gì và cách viết chuẩn SEO

+ SEO là search engine optimize là công cụ tối ưu hoá việc tìm kiếm cho trang web trên các nền tảng google,bing.v.v…

+ Cách viết chuẩn SEO bao gồm việc sử dụng các thẻ semantic, từ khóa, mô tả và các thẻ meta để cung cấp thông tin cho các công cụ tìm kiếm và tối ưu hóa nội dung trang web.

- Có bao nhiêu cách responsive

+ Có 2 cách responsive là **Media Queries (Câu lệnh truy vấn tới màn hình) từ 768px-1024px là tablet , Fluid Grids (Layout lưới 12 cột).**

JS / TS :

- Phân biệt const,let,var

+ const : Biến được khai báo bằng `const` là biến không thể thay đổi giá trị sau khi gán giá trị ban đầu. Phạm vi của biến `const` có thể là toàn cục (nếu được khai báo ở mức ngoài các hàm) hoặc cục bộ (nếu được khai báo trong hàm).

+ let : Biến được khai báo bằng `let` là biến có thể thay đổi giá trị sau khi đã được gán giá trị ban đầu và không thể khai báo lại. Phạm vi của biến `let` có thể là toàn cục (nếu được khai báo ở mức ngoài các hàm) hoặc cục bộ (nếu được khai báo trong hàm).

+ var : Biến được khai báo bằng `var` cũng có thể thay đổi giá trị sau khi đã được gán giá trị ban đầu.Tuy nhiên, biến `var` có phạm vi hoisting (nâng biến), nghĩa là biến được "nâng" lên đầu của phạm vi của nó trước khi chương trình chạy.Tránh sử dụng var vì khó kiểm soát khi chương trình lớn.

- Tính chất hoisting là gì ? Là tính chất mà js sẽ quét qua toàn bộ source kiểm tra xem đã khai báo chưa, điều này có nghĩa là bạn có thể truy cập các biến và hàm trước khi chúng thực sự được khai báo trong mã nguồn. Và chỉ áp dụng cho từ khoá `var` và declarations function (func thường)

- Deadzone là gì ? "Dead zone" trong ngữ cảnh của JavaScript là một khái niệm liên quan đến phạm vi và hoisting (nâng biến) của biến. Dead zone (vùng chết) là một phần của mã nguồn trong đó biến đã được khai báo bằng `let` hoặc `const` tồn tại, nhưng chưa được khởi tạo và không thể truy cập. Dead zone giúp ngăn chặn việc truy cập biến trước khi nó đã được khởi tạo, giúp ngăn lỗi logic và tạo ra hành vi rõ ràng hơn trong việc quản lý biến.

```
console.log(a); // Error: "a is not defined"
let a = 10;
```

- Tham chiếu , tham trị là gì ?

  + Tham trị (các kiểu dữ liệu nguyên thuỷ) : là khi mình trỏ tới 1 biến có kiểu dữ liệu tham trị nó sẽ copy giá trị của biến đó. Các kiểu dữ liệu nguyên thủy bao gồm: số, chuỗi, boolean, null, undefined.

  + Tham chiếu (các kiểu dữ liệu đặc biệt) : là khi mình trỏ tới các biến có kiểu dữ liệu tham chiếu thì mình sẽ tham chiếu đến địa chỉ vùng nhớ của biến đó và sẽ

cùng nhận giá trị của địa chỉ vùng nhớ đó. Các kiểu dữ liệu tham chiếu bao gồm: array , obj , function , class.

- HOF là gì ?

  + HOF là viết tắt của "Higher-Order Function" trong lập trình. Một Higher-Order Function là một loại hàm trong ngôn ngữ lập trình hướng hàm (như JavaScript) có khả năng nhận một hàm khác làm tham số hoặc trả về một hàm khác như kết quả.

  + Higher-Order Function có thể thực hiện ít nhất một trong các hoạt động sau:

  1. **Nhận một hàm làm tham số**: Hàm này có thể thực thi hoặc xử lý hàm được truyền vào như một tham số.
  2. **Trả về một hàm khác**: Hàm này có thể tạo và trả về một hàm mới.

- Pure Function là gì ?

  + **Pure function** là hàm luôn trả về cùng một kết quả khi tham số đầu vào không thay đổi, cho dù hàm đó được thực thi bao nhiêu lần đi chăng nữa. Giá trị mà pure function trả về chỉ phụ thuộc duy nhất vào giá trị tham số truyền vào, nó không phụ thuộc vào bất cứ trạng thái, dữ liệu bên ngoài phạm vi của hàm.

```
// Hàm thuần khiết
function add(a, b) {
    return a + b;
}
const result = add(3, 5); // Kết quả: 8
```

==========

```
//Hàm không thuần khiết vì kết quả trả về có tác động từ bên ngoài bởi biến tax
let tax = Math.random(100);
// Hàm tính tổng tiền sau thuế
function sumAfterTax(a, b) {
    return  ((a + b) * (tax / 100)).toFixed(0)
}
console.log(sumAfterTax(1000, 9000)) // output: 43
```

- Phân biệt null vs undefined

+ `null` là 1 là một giá trị được gán cho biến để biểu thị rằng biến có giá trị không tồn tại hoặc chưa được khởi tạo , `undefined` là kiểu dữ liệu mặc định khi mà chỉ khai báo biến mà chưa gán giá trị

- Cơ chế bất đồng bộ là gì, event loop và các cách xử lí bất đồng bộ

  +Trong JS là ngôn ngữ đơn luồng, thực hiện từ trên xuống dưới cái nào viết trước chạy trước , cơ chế bất đồng bộ (asynchronous) trong JS liên quan đến khả năng thực hiện nhiều tác vụ cùng một lúc mà không chặn luồng chính của chương trình.

  +Event loop là Các tác vụ đồng bộ được thực hiện trực tiếp trên luồng chính, còn các tác vụ bất đồng bộ được đưa vào hàng đợi sự kiện (callback queue) để chờ thực hiện sau khi luồng chính chạy xong. Khi luồng chính hoàn thành việc thực hiện, Event Loop sẽ lấy tác vụ từ callback queue và thực hiện chúng. Quá trình này diễn ra liên tục.

  +Có 3 cách để xử lí bất đồng bộ : Callback , Promise và async / await (của ES7)

```
// Cách xử lí vs Callback
function fetchData(callback) {
  setTimeout(function() {
    const data = { message: "Hello, world!" };
    callback(data);
  }, 1000);
}

function handleData(data) {
  console.log(data.message);
}

fetchData(handleData);
```

```
//Cách xử lí Promise
function fetchData() {
    return new Promise((resolve, reject) => {
        // Thực hiện tác vụ bất đồng bộ
        setTimeout(() => {
            resolve("d"); // Hoàn thành thành công
        }, 1000);
    });
}
```

```
fetchData()
    .then((result) => {
        console.log(result)
    })
    .then(()=>{
        console.log("a")
        console.log("b")
        console.log("c")
    })
    .catch((error) => {
        console.error(error);
    });
```

```
//Cách xử lí vs async await
async function fetchData() {
    // Thực hiện tác vụ bất đồng bộ
    return new Promise((resolve) => {
      setTimeout(() => {
        const data = "Dữ liệu đã được tải";
        resolve(data);
      }, 1000);
    });
  }

  async function main() {
    const result = await fetchData();
    console.log(result); // Dữ liệu đã được tải
  }
  main();
```

- Các method của Array ,Object

  +Tự nhớ…

- Phân biệt arrow func vs func thường (expression vs declaration)

+ Arrow function (hàm mũi tên) và function thường (còn gọi là function expression hoặc function declaration) là hai cách để định nghĩa hàm trong JS. Có cú pháp ngắn gọn hơn so với function thường , Arrow function không có context riêng (không có `this`)

và sẽ tham chiếu đến `this` của phạm vi bên ngoài và không nên định nghĩa nó trong obj || class và không có hoisting. Không thể sử dụng `arguments` **trong arrow func.**

- Phân biệt toán tử spread vs rest

  +Toán tử spread được sử dụng để copy mảng và obj.

  +Toán tử rest được sử dụng trong khai báo hàm để lấy tất cả các tham số còn lại và đưa chúng vào một mảng. Điều này giúp xử lý số lượng tham số không biết trước.

- 4 tính chất OOP,4 tính chất đó thể hiện qua những cú pháp nào trong typescirpt
  - Tính đóng gói: là tính chất dùng để bảo vệ thuộc tính và phương thức không thể bị tác động từ các logic từ bên ngoài. Cú pháp để thể hiện tính đóng gói là `private`
  - Tính trừu tượng: là sẽ ẩn dấu đi những chi tiết không cần thiết của 1 đối tượng , quá trình tạo ra một lớp cơ sở với các đặc điểm chung và sau đó xác định các lớp con từ lớp cơ sở đó. Cú pháp để thể hiện tính trừu tượng là `abstract`
  - Tính kế thừa: là tính chất cho phép 1 lớp con có thể kế thừa thuộc tính và phương thức của lớp cha . Cú pháp thể hiện tính kế thừa là extends ⇒ emplement không phải cú pháp thể hiện tính kế thừa mà chỉ để triển khai 1 interface.
  - Tính đa hình: là tính chất cùng 1 phương thức kế thừa từ lớp cha có thể có nhiều cách thực hiện khác nhau trong lớp con, cú pháp thể hiện qua việc ghi đè các phương thức khi kế thừa từ lớp cha

- Phân biệt type annotation vs inference

+ **Type Annotation (Chú thích kiểu)** xác định một cách rõ ràng kiểu dữ liệu mà biến hoặc tham số phải có lúc nó được định nghĩa. `let a:number = 1`

+ **Type Inference (Suy luận kiểu) là** khả năng của TypeScript tự động xác định kiểu dữ liệu của biến dựa trên giá trị mà bạn gán cho nó và sau đó sử dụng kiểu đó cho biến đó.

```
let a = 1 // a là kiểu number
```

- Các kiểu dữ liệu đặc biệt trong TS

  + any : Kiểu dữ liệu `any` biểu diễn bất kỳ kiểu dữ liệu nào.

  + unknown : Kiểu dữ liệu `unknown` tương tự như `any`, nhưng an toàn hơn. Bạn không thể thực hiện các phép toán trên kiểu `unknown` trực tiếp nhưng cần thực hiện kiểm tra kiểu trước khi sử dụng.

  + void : Kiểu dữ liệu `void` được sử dụng cho các hàm không trả về giá trị nào (hoặc trả về `undefined`).

  + never : Kiểu dữ liệu `never` biểu diễn các giá trị mà không thể xảy ra. Ví dụ, một hàm sẽ throw exception hoặc bị vòng lặp vô hạn.

  + Tuple : Kiểu dữ liệu `tuple` biểu diễn một mảng có số lượng và kiểu phần tử cố định, nhưng các phần tử không cần phải cùng kiểu.

  + enum : Kiểu dữ liệu `enum` được sử dụng để tạo ra 1 danh sách,được xác định trước.

  + union : Kiểu dữ liệu `union` cho phép bạn kết hợp nhiều kiểu dữ liệu lại với nhau. Ví dụ, `number | string` biểu diễn giá trị có thể là số hoặc chuỗi.

  + intersection type : Kiểu dữ liệu `intersection` cho phép bạn kết hợp các kiểu dữ liệu lại với nhau để tạo ra một kiểu mới chứa các thuộc tính và phương thức của cả hai kiểu bằng cú pháp &.

- Phân biệt abstract class vs class thường

  + abstract class thì sẽ giống như 1 class thường nhưng có thể định nghĩa các abstract method. Khi class con kế thừa từ abstract class sẽ bắt buộc phải thực thi abstract method

  + không thể tạo một đối tượng từ abstract class bằng từ khoá new như class thường. Nó chỉ có thể được kế thừa bởi các class con.

  + abstract class con kế thừa abstract class cha thì k cần thiết phải triển khai abstract method.

- Từ khoá super trong class

+ Được sử dụng để tham chiếu đến class cha khi 1 class con kế thừa :

- **Constructor của lớp con:**
  Khi bạn tạo một constructor trong lớp con, bạn có thể gọi constructor của lớp cha bằng cách sử dụng `super()`. Điều này giúp bạn chuyển các đối số hoặc thực hiện các tác vụ khởi tạo từ lớp cha trước khi thực hiện các tác vụ của lớp con.
- **Gọi phương thức của lớp cha:**
  Bạn có thể sử dụng `super` để gọi các phương thức của lớp cha từ lớp con. Điều này hữu ích khi bạn muốn sử dụng lại các logic từ lớp cha hoặc mở rộng chúng trong lớp con.
- **Truy cập thuộc tính của lớp cha:**
  Bạn có thể truy cập các thuộc tính của lớp cha thông qua `super` để lấy giá trị của chúng hoặc thậm chí gán giá trị mới cho chúng.

- JSON là gì

  + JSON là viết tắt của "JavaScript Object Notation". Đó là một định dạng dữ liệu được sử dụng để truyền tải và lưu trữ dữ liệu trong các ứng dụng. JSON không phải là một ngôn ngữ lập trình, mà chỉ là một cách biểu diễn dữ liệu dưới dạng văn bản. JSON thường được sử dụng trong các ứng dụng web để giao tiếp giữa máy khách (client) và máy chủ (server) hoặc để lưu trữ dữ liệu trong cơ sở dữ liệu.

ReactJS :

- Liệt kê các hook và mô tả nó

  + **useState:** Hook này cho phép bạn thêm trạng thái (state) cục bộ vào thành phần hàm của bạn. Khi trạng thái thay đổi, React sẽ tự động render lại thành phần.

  + **useEffect**: cho phép bạn thực hiện các tác vụ sau khi render hoàn thành. Điều này có thể là sự tương tác với DOM, gọi API, đăng ký sự kiện, vv. Nó thay thế các lifecycle methods trong các thành phần dựa trên lớp.
  + **useContext:** cho phép bạn truy cập vào các giá trị trong context của component cha, giúp truyền dữ liệu qua nhiều cấp thành phần mà không cần truyền qua các props.

+ **useReducer:** tương tự như `useState`, nhưng thay vì quản lý một giá trị đơn, nó quản lý một trạng thái phức tạp hơn thông qua việc sử dụng reducer function.

+ **useMemo:** giúp bạn tối ưu hóa hiệu suất bằng cách lưu giữ kết quả của một hàm tính toán giữa các lần render liên tiếp. useMemo sẽ luôn trả về 1 kết quả.

+ **useCallback:** giúp bạn tối ưu hóa hiệu suất bằng cách lưu giữ phiên bản của một hàm callback giữa các lần render liên tiếp.

+ **useRef** : giúp bạn truy cập các tham chiếu đến các phần tử DOM hoặc lưu giữ giá trị mà không gây render lại.

- Virtual DOM là gì và vì sao nó nhanh hơn DOM thật

  + Virtual DOM (ViDOM) là một khái niệm trong thư viện React (và một số thư viện khác) để tăng hiệu suất khi cập nhật giao diện người dùng. ViDOM không phải là một phần của trình duyệt, mà là một cấu trúc dữ liệu trong bộ nhớ của ứng dụng. Mục tiêu chính của ViDOM là làm cho việc cập nhật giao diện hiệu quả hơn bằng cách giảm số lượng thao tác trực tiếp trên DOM thật.

  + Virtual DOM sao lại nhanh hơn DOM thật :

  1. **Giảm tương tác với DOM thật:** So với việc cập nhật trực tiếp trên DOM, Virtual DOM chỉ cập nhật những phần thay đổi thực sự, giảm tương tác với DOM thật.
  2. **Chuyển đổi hiệu quả hơn:** Quá trình so sánh và tính toán sự thay đổi trên Virtual DOM giúp React chỉ áp dụng những thay đổi cần thiết lên DOM thật, tránh việc làm cho toàn bộ DOM phải cập nhật mỗi khi trạng thái thay đổi.
  3. **Minh bạch hóa quá trình cập nhật:** ViDOM giúp làm rõ quá trình cập nhật giao diện, giúp các lập trình viên dễ dàng hiểu và kiểm soát việc cập nhật.

- Phân biệt props , state

  + props (properties) là gía trị được truyền từ component cha xuống cho component con.Component con chỉ có thể nhận và sử dụng giá trị từ props chứ k thể thay đổi props.

  + state là dữ liệu được quản lý bởi chính component đó tạo ra và có thể thay đổi trong suốt thời gian hoạt động của ứng dụng , thường được sử dụng để lưu trữ

trạng thái nội bộ của một thành phần và có thể bị thay đổi bởi các sự kiện hoặc tương tác người dùng.

+ Khi props ,state thay đổi thì component sẽ được re-render.

- Cơ chế hoạt động của useState

```
const [state,setState] = useState(initValue)
```

+ Dùng để quản lí các trạng thái,dữ liệu tại cục bộ của component và có thể thay đổi trong quá trình sử dụng ứng dụng tại component đó.

+ Khi state thay đổi bằng setState() thì component sẽ được re-render lại để cập nhật UI theo dữ liệu hoặc trạng thái vừa thay đổi.

+ Khi setState thì nó sẽ k set lại ngay lập tức mà lên lịch cho việc re-render UI cho đến khi k còn tác vụ nào thì mới setState, thay vì cập nhật ngay lập tức, React có thể sẽ chờ và thực hiện việc cập nhật trạng thái và render lại giao diện theo một thời điểm tối ưu hóa.

- Cơ chế hoạt động của useEffect

```
useEffect(()⇒{
    //logic code
},[ dependencies ])
```

+ useEffect sẽ được chạy ngay sau khi UI đc render nhằm tối ưu cho việc ưu tiên UI trước rồi mới thực hiện các thao tác side effect.

+ Sau lần render đầu tiên useEffect sẽ được chạy lại theo giá trị dependencies truyền vào , nếu là mảng rỗng thì useEffect chỉ chạy lần đầu , nếu có dependencies thì useEffect sẽ chạy lần đầu và mỗi khi dependencies thay đổi, còn nếu không truyền mảng dependencies vào thì sẽ bị chạy liên tục khi re-render (Không sử dụng).

- Side effect là gì ?

+ Side-effect là một khái niệm chỉ các thay đổi hoặc tác động mà một hàm hoặc một phần của chương trình gây ra bên ngoài phạm vi của nó. Thường sử dụng useEffect để xử lí các side-effect.

Ví dụ một số side effects hay gặp:

- Tạo HTTP request tới API (AJAX/fetch)
- Ghi – Cập nhật dữ liệu vào file
- Lưu dữ liệu vào Database, Local Storage
- Hiển thị dữ liệu ra màn hình
- Thay đổi nội dung của DOM
- vân vân và mây mây…

- Lifecyle của React

  + Các lifecycle trong ReactJS :

    - **componentDidMount :** component đã được thêm vào DOM mounting và render lần đầu tiên

    - **componentDidUpdate :** được gọi sau khi component đã được cập nhật (props và state thay đổi) và re-render lại.

    - **componentWillUnmount :** được gọi trước khi component bị huỷ bỏ (unmounting) khỏi DOM. Đây là nơi bạn có thể dọn dẹp các tài nguyên không còn cần thiết như đăng ký sự kiện hoặc hủy các tác vụ đang chạy.

  + Các lifecycle này có thể đc hook useEffect thay thế.

- Khác biệt giữa func component vs class component và tại sao sử dụng func component để thay thế

  + ClassComponent : Cú pháp sử dụng class , có các method lifecycle để quản lí vòng đời của component.

  + FunctionComponent : Cú pháp viết dưới dạng function , sử dụng hooks.

  + Nên sử dụng funcComponent vì cú pháp ngắn gọn , đơn giản , tối ưu về hiệu suất hơn so vs class component ,được cung cấp react hook nên đầy đủ mọi tính năng như class component cũ.

- Redux là gì,cơ chế hoạt động của redux

  + Redux là một thư viện quản lý trạng thái toàn cục (state management) , giúp thực hiện việc quản lí dữ liệu ,trạng thái trong những dự án lớn dễ dàng.

  + Cơ chế hoạt động của Redux:

  1. **Store (Kho lưu trữ):** Redux sử dụng một kho lưu trữ duy nhất (store) để lưu trữ toàn bộ trạng thái của ứng dụng. Kho lưu trữ này được tạo ra bằng cách sử dụng hàm `createStore` và chứa dữ liệu cần được quản lý.
  2. **Actions (Hành động):** Actions là các sự kiện mô tả việc thay đổi trạng thái của ứng dụng. Mỗi action đều phải có một loại (type) và có thể đi kèm với dữ liệu (payload) cần được cập nhật.
  3. **Reducers (Bộ Reducer):** Reducers là các hàm xử lý việc thay đổi trạng thái dựa trên các hành động. Mỗi reducer nhận vào một trạng thái hiện tại và một hành động, sau đó trả về trạng thái mới.
  4. **Dispatch (Phát đi):** Bằng cách gọi hàm `dispatch`, bạn gửi một actions vào store. Store sẽ truyền hành động này tới các reducer để cập nhật trạng thái.
  5. **Subscribe (Theo dõi):** Bạn có thể đăng ký (subscribe) cho store để theo dõi bất kỳ thay đổi nào trong trạng thái. Khi trạng thái thay đổi, các hàm callback đã đăng ký sẽ được gọi để cập nhật giao diện.

- Redux toolkit là gì và vì sao sử dụng.

  + Redux Toolkit là một bộ công cụ được cung cấp bởi Redux để giúp đơn giản hóa và tối ưu hóa việc sử dụng Redux trong ứng dụng React.Nó còn cung cấp việc xử lí bất đồng bộ trong redux.

  + Lý do sử dụng Redux Toolkit:

  - **Giảm Boilerplate Code:** Redux Toolkit giúp giảm bớt mức độ phức tạp và lượng mã lặp lại (boilerplate) khi viết Redux trong ứng dụng. Việc tạo action, reducer và selector trở nên ngắn gọn hơn và dễ dàng hơn.
  - **Tích Hợp Thông Minh:** Redux Toolkit cung cấp các cài đặt mặc định thông minh cho việc tạo store, cải thiện trải nghiệm làm việc với Redux.
  - Cung cấp createAsyncThunk để xử lí bất đồng bộ vs redux.

- Phân biệt giữa ServerSideRendering vs ClientSideRendering

  + **Server-Side Rendering (SSR):**

  1. Tải tài nguyên theo request
  2. Tối ưu SEO ,các công cụ tìm kiếm như Google có thể dễ dàng tìm thấy và chỉ mục nội dung trên trang web, do đó cải thiện khả năng tìm kiếm (SEO).
  3. Làm load lại trang
  4. Ko bảo mật trang vì có thể xem được source code ,làm các trang thương mại,trang báo…

  + **Client-Side Rendering (CSR):**

  1. **Tải Nhanh Trang Sau Khi Đã Được Tải Lần Đầu:** Nặng trong lần tải đầu vì phải lấy toàn bộ tài nguyên trong lần tải đầu .
  2. **Tương tác Nhanh Hơn:** Vì phần lớn công việc tải dữ liệu và cập nhật trang diễn ra trên trình duyệt và ko load lại trang.
  3. **Phụ Thuộc Nhiều vào JavaScript:** CSR yêu cầu JavaScript phải hoạt động một cách tốt để trình duyệt có thể hiển thị nội dung. Nếu JavaScript không được hỗ trợ hoặc có lỗi, trang có thể không hiển thị đúng hoặc không hoạt động.
  4. Bảo mật trang,thường làm các trang admin,các trang cần bảo mật.
  5. **SEO Khó Khăn Hơn:** Một số công cụ tìm kiếm có thể gặp khó khăn trong việc hiểu và chỉ mục nội dung được tạo bởi JavaScript, dẫn đến vấn đề về SEO.

SQL :

- Phân biệt khoá chính,khoá ngoại

  + Khoá chính (PK): là một cột trong bảng cơ sở dữ liệu mà giá trị của nó duy nhất định và không được để trống , đảm bảo tính duy nhất và xác định dòng dữ liệu.Mỗi bảng chỉ có thể có 1 khoá chính.

  + Khoá ngoại (FK): là một cột trong một bảng cơ sở dữ liệu tham chiếu đến khoá chính của một bảng khác.Khi sử dụng khoá ngoại, bạn có thể thực hiện các thao

tác JOIN để truy vấn dữ liệu từ nhiều bảng cùng lúc.

- Thứ tự hoạt động các câu truy vấn trong Mysql

  1. **FROM:** Đây là bước đầu tiên trong một câu truy vấn. Nó chỉ định bảng sẽ được truy vấn.
  2. **JOIN:** Nếu bạn sử dụng các thao tác JOIN để kết hợp dữ liệu từ nhiều bảng, phần này sẽ thực hiện các bước kết hợp dữ liệu từ các bảng theo các điều kiện được xác định.
  3. **WHERE:** Bước này áp dụng điều kiện lọc dữ liệu dựa trên các điều kiện được xác định trong phần WHERE. Các dòng không thỏa mãn điều kiện này sẽ bị loại bỏ.
  4. **GROUP BY:** Nếu bạn sử dụng GROUP BY để nhóm các dòng dữ liệu dựa trên một hoặc nhiều cột, phần này thực hiện việc nhóm dữ liệu lại.
  5. **HAVING:** Nếu bạn sử dụng HAVING để lọc dữ liệu đã được nhóm dựa trên các điều kiện, phần này thực hiện lọc dữ liệu sau khi đã nhóm.
  6. **SELECT:** Bước này chọn các cột cụ thể từ kết quả dữ liệu sau các bước trước.
  7. **DISTINCT:** Nếu bạn sử dụng DISTINCT để chỉ lấy các giá trị duy nhất, phần này sẽ loại bỏ các giá trị trùng lặp.
  8. **ORDER BY:** Nếu bạn sử dụng ORDER BY để sắp xếp dữ liệu theo một hoặc nhiều cột, phần này thực hiện việc sắp xếp.
  9. **LIMIT:** Bước cuối cùng xác định số lượng kết quả dữ liệu tối đa mà bạn muốn trả về.

- Các kiểu qh trong sql

  + 1-1 : Là 1 trường của bảng A chỉ liên kết vs 1 trường của bảng B và ngược lại.

  + 1-n : Là 1 trường của bảng A có thể liên kết vs nhiều của bảng B ,nhưng mỗi trường của bảng B chỉ liên kết vs 1 trường của bảng A.

  + n-n : Là 1 trường của bảng A có thể liên kết vs nhiều trường của bảng B và ngược lại. Để biểu diễn quan hệ này, thường sử dụng một bảng trung gian (table) để kết

nối hai bảng chính.

- Câu lệnh khởi tạo,chỉnh sửa,xoá,chèn dữ liệu trong table

  + Tự nhớ….

- JOIN là gì và các kiểu JOIN bảng trong sql

  + JOIN là thực hiện việc nối 2 bảng theo chiều ngang.Có 3 kiểu JOIN trong mysql, điều kiện khi join bảng sẽ sử dụng mệnh đề ON.

  + Inner join là trả về các bản ghi chỉ khi có sự khớp trong tất cả điều kiện JOIN giữa các bảng.

```
SELECT *
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

  + Left join là trả về tất cả các bản ghi từ bảng bên trái (left table,các trường k khớp sẽ bằng Null) và các bản ghi khớp từ bảng bên phải (right table) dựa trên điều kiện JOIN.

```
SELECT *
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

  + Right join Trả về tất cả các bản ghi từ bảng bên phải (right table,các trường k khớp sẽ bằng Null) và các bản ghi khớp từ bảng bên trái (left table) dựa trên điều kiện JOIN.

```
SELECT *
FROM Customers
RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

  + Có thể full join giữa 2 bảng bằng cách dùng UNION giữa câu 2 câu lệnh query select right join và left join .

- Index là gì

  link tham khảo: https://www.youtube.com/watch?v=4bDRJSnXUAs

  + index là một cơ chế tối ưu hóa cơ sở dữ liệu được sử dụng để tăng tốc độ truy vấn và tìm kiếm dữ liệu. Index là một cấu trúc dữ liệu đặc biệt được tạo ra từ một

hoặc nhiều cột trong bảng để cho phép cơ sở dữ liệu tìm kiếm và truy cập dữ liệu một cách nhanh chóng hơn

+Đánh index trên các cột của bảng
+ Tăng tốc độ truy vấn
+ Giảm độ thao tác dữ liệu
+ Sắp xếp dữ liệu

+ có hai cách để đánh index
-- c1: dùng ALTER TABLE

```
ALTER TABLE tableName ADD UNIQUE INDEX  nameIndex(nameColumn) //thêm index
ALTER TABLE tableName DROP INDEX nameIndex //xoá index
```

-- c2: dùng create để đánh index

```
CREATE INDEX nameIndex ON tableName(name DESC) //thêm index
DROP INDEX nameIndex ON tableName //xoá index
```

- Có 2 loại index là UNIQUE INDEX vs INDEX: UNIQUE INDEX thì cột đó dữ liệu sẽ không được trùng, còn INDEX thì dữ liệu được trùng.

Node / ExpressJS :

- Mô hình MVC là gì ?

  + Mô hình MVC (Model-View-Controller) là một kiến trúc phổ biến được sử dụng trong phát triển phần mềm, đặc biệt là trong phát triển ứng dụng web.

  - Model: Sẽ đại diện cho database và thực hiện các logic với database.
  - View: Sẽ đại diện cho phần UI ,tương tác vs người dùng.
  - Controller: là bộ điều khiển của ứng dụng, điều phối việc giao tiếp giữa model và view, controller thực hiện các xử lý logic và quyết định nghiệp vụ.

- HTTP là gì và các phương thức HTTP

  + HTTP (Hypertext Transfer Protocol) là một giao thức truyền thông dựa trên request-response giữa máy khách (client) và máy chủ (server) trên mạng. Nó được sử dụng để truyền tải các tài liệu siêu văn bản (như trang web) và dữ liệu liên quan trên Internet.

  + Các phương thức HTTP :

1. **GET:** Phương thức GET được sử dụng để yêu cầu dữ liệu từ máy chủ.
2. **POST:** Phương thức POST được sử dụng để gửi dữ liệu đến máy chủ để xử lý.
3. **PUT:** Phương thức PUT được sử dụng để cập nhật dữ liệu của tài nguyên hoặc tạo mới tài nguyên nếu chưa tồn tại.
4. **DELETE:** Phương thức DELETE được sử dụng để xóa tài nguyên khỏi máy chủ. Nó thực hiện xóa tài nguyên được xác định trong yêu cầu.
5. **PATCH:** Phương thức PATCH được sử dụng để thực hiện một hoặc nhiều thay đổi nhỏ vào tài nguyên.
6. **HEAD:** Phương thức HEAD tương tự như GET, nhưng nó chỉ yêu cầu thông tin về phản hồi mà không cần phần thân của phản hồi. Nó thường được sử dụng để kiểm tra trạng thái hoặc thông tin của tài nguyên mà không cần tải toàn bộ dữ liệu.
7. **OPTIONS:** Phương thức OPTIONS yêu cầu thông tin về các phương thức HTTP mà máy chủ hỗ trợ cho một tài nguyên cụ thể. Điều này cho phép máy khách biết được các phương thức mà nó có thể sử dụng để tương tác với tài nguyên.

- Phân biệt GET vs POST

  + GET : Dữ liệu được gửi đi sẽ hiển thị trên thanh địa chỉ của trình duyệt, bảo mật kém , giới hạn kí tự , nhanh hơn POST vì được cache,nơi chứa dữ liệu params.

  + POST : Dữ liệu gửi đi không hiển thị trên thanh địa chỉ của trình duyệt, an toàn hơn GET, không giới hạn kí tự gửi lên , nơi chứa dữ liệu là body.

- Phân biệt PUT vs PATCH

  + PUT : Phương thức PUT được sử dụng để cập nhật toàn bộ dữ liệu. Điều này có nghĩa rằng dữ liệu cũ sẽ bị ghi đè bởi dữ liệu mới.Khi sử dụng PUT, bạn cần cung cấp toàn bộ dữ liệu mới cho tài nguyên đó.

  + PATCH : Phương thức PATCH được sử dụng để cập nhật một phần của tài nguyên.Bạn chỉ cần cung cấp các trường dữ liệu bạn muốn cập nhật, không cần gửi toàn bộ tài nguyên.

- Middleware là gì

  + Middleware là các bước trung gian giúp kiểm tra, sửa đổi hoặc tạo ra dữ liệu để đảm bảo rằng quá trình xử lý yêu cầu được thực hiện một cách chính xác và hiệu quả.

  + Có 2 cách sử dụng middleware :

    - Sử dụng middleware global cho toàn bộ ứng dụng app.
    - Sử dụng middleware ở router để kiểm tra trước khi vào controller.

- Authen vs Author là gì ?

  + **Authentication (Xác thực): là quá trình xác định và xác minh danh tính của người dùng. Quá trình xác thực thường dựa trên thông tin xác định như tên đăng nhập và mật khẩu.**

  + **Authorization (Phân quyền): là quá trình quyết định xem người dùng có được phép truy cập vào các tài nguyên, chức năng hoặc hành động hay không. Điều này thường dựa trên các quyền và vai trò đã được định nghĩa trước trong database.**

- Các thư viện hay sử dụng trong dự án BE và nói về mục đích sử dụng nó

  + Tự nhớ….

- Template engine là gì

  + Template engine là một công cụ giúp tạo ra nội dung động dựa trên các mẫu (templates) đã định nghĩa trước. Nó giúp render giao diện ở phía server từ đó có thể tạo ra 1 ứng dụng web vs cơ chế server side rendering.

- Restful API là gì

  + RESTful API (Representational State Transfer) là một kiến trúc và một cách tiếp cận trong thiết kế các giao diện lập trình ứng dụng (viết API) cho các ứng dụng

web. Nó thường được sử dụng để xây dựng các dịch vụ web có khả năng mở rộng, dễ dàng quản lý và tương tác với các ứng dụng khác thông qua giao thức HTTP.

- Router trong expressjs là gì

  + Router là một cơ chế giúp bạn tổ chức mã nguồn của ứng dụng web thành các phần nhỏ hơn, dễ quản lý hơn. Mỗi phần nhỏ này tương ứng với một tập hợp các tuyến đường (routes) liên quan đến chức năng hoặc phần của ứng dụng cụ thể. Điều này giúp mã nguồn trở nên rõ ràng hơn và dễ bảo trì khi ứng dụng phức tạp.

  ========== huudongRA ==========