1. **Redux thunk:**

Redux-Thunk là một middleware được dùng để xử lý các action bất đồng bộ trong Redux.



Redux: Action trả về 1 plain object => muốn trả về object sau 1 khoảng thời gian => redux thunk => cho phép trả về 1 function (dispatch), nếu trả về là object thi THUNK sẽ chuyển về reducer như thường lệ, nếu trả về 1 function, THUNK sẽ đợi action đó hoàn thành và đưa kết quả reducer

Redux-Reducer không được có side effect, chỉ phụ thuộc state và action, không run logic bất đồng bộ (ajax, timeout, promise)

1. **Toolkit**

(1 thu vien setup san 1 so thu trong redux, giup viet redux ngan hon, tot hon, de hon)

**createAsyncThunk** chỉ giúp tạo ra một luồng hoạt động bất đồng bộ và**dispatch action**

**configStore: setup store (**tich hop redux devtool va redux thunk)

**createReducer: cho phép tạo ra 1 reducer 1 cách đơn giản , cho phép handle mutate data, tự handle case default -** đầu vào là 1 state và 1 object giúp reducer lắng nghe các action và update state

**createSlice tao ra 1 slice bao gom name, reducer va initial state**

reducer: nhận action và update lai các state

extrareducers: là các reducers lắng nghe các action từ nơi khác, kể cả action từ slice khác hoặc từ action thunk

**createAction: tạo ra 1 action một cách ngắn gon, nhận vào 1 string và trả ra 1 action có type là string vừa truyền vào**

1. **Primitive data type và reference type**

Primitive data type: (kiểu nguyên thủy – chứa trực tiếp giá trị) là khi gán nó cho một biến khác, một bản sao của nó đc tạo ra và gán cho biến mới (khác địa chỉ) *– Number, String, Boolean, null, and undefined*.

Reference type: (kiểu tham chiếu – nó chứa địa chỉ) là khi gán cho một biến khác thì không tạo ra bản sao, 2 biến có cùng địa chỉ - *Array, Object, and Function.*

Chỉ lưu địa chỉ nơi chứa giá trị

phép gán (=) với oject chính là copy địa chỉ

1. **React Context**

Là cách thức để tạo ra props toàn cục và các comp có thể nhận nó một cách đơn giản, tránh drilling props

*Ưu điểm*:

* component con có thể nhận props một cách đơn giản
* Tránh code lặp lại, tránh bug k mong muốn, dễ maintain
* Thích hợp với app nhỏ

*Nhược điểm*:

* component cha không thể nhận props từ comp con
* không thể truyền data cho các comp khác (khác quan hệ cha con)
* khó quản lý và maintain trong app lớn
* chuyển dùng redux

1. **Reselect**

* Là một thư viện tạo bộ nhớ cho các bộ chọn trong redux
* Nó sẽ ghi nhớ lại kết quả cùng với 1 bộ input, nếu lần tiếp theo cùng bộ input thì sẽ không tính toán mà sẽ trả ra kết quả ngay
* Hạn chế việc rerender không cần thiết trong redux

1. **Real Dom >< Vitural Dom (tăng hiệu xuất web)**

*Cấu trúc trang web hiên nay khá phức tạp nên cây DOM khá lớn => update & query DOM rất nhiều => vấn đề hiệu năng => Virtual DOM*

***DOM*** *là Document Object Model một web API interface cho HTML và XML, nó định nghĩa cấu trúc của document và cách document đc truy cập để nó thay đổi cấu trúc, style và nội dung*

*DOM ảo: là 1 bản sao của DOM*

* Là đại diện của 1 DOM thật đc lưu trong bộ nhớ và đồng bộ với DOM thật
* đc update thông qua API của React
* Tìm và update DOM nhanh hơn bởi thuật toán diffing
* Không lãng phí bộ nhớ

*DOM thật:*

* đc update thông qua DOM api

các thay đổi trên UI đc đưa vào V.DOM => sử dụng diffing để tìm những thay đổi đang diễn ra => update những phần tử đã thay đổi, bỏ qua những chỗ không thay đổi

1. **Props vs State**

Props là tham số đầu vào của 1 component, comp không thể thay đổi nó

State là trạng thái của 1 component, phạm vi được giới hạn trong comp, comp có thể cập nhật nó

1. **Class Comp (stateful) >< Function Comp (stateless)**

* Function comp là 1 comp đơn giản hơn, nhận đối số là props và trả về 1 react element

Không bị rang buộc bởi THIS

Dễ viết, ngắn gọn, đơn giản

Dùng khi không cần đến state

Loại bỏ constructor, render

Các life cycle đc đơn giản hóa bằng useEffect

Quản lý state đơn giản hơn với react hook

* Class Comp là 1 class được kế thừa từ react.component, nó có các thuộc tính và phương thức của react.component

1. **Ưu điểm React, tính năng chính (hoạt động ntn)**

React creates a VIRTUAL DOM in memory.

React only changes what needs to be changed!

Chia nhỏ các thành phần phức tạp thành comp và tái sử dụng

1. **Error Boundary**

Là 1 comp bắt lỗi javascript ở bất kỳ đâu, nếu error bắt đc lỗi, đâu tiên sẽ log lỗi và hiển thị UI thông báo thay vì cash cả app

Nó bắt lỗi ở quá trình render, life cycle và constructor

*Limitation*: không bắt được

Lỗi sự kiện

Lỗi code bất đồng bộ

Server side rendering

Lỗi từ chính Boudary Comp

1. **HOC**

Là 1 function nhận vào 1 comp và trả ra 1 comp (vi du Redux connect, withRouter)

Dùng để chia sẽ function chung giữa các component, giảm lặp lại phần code chung

Biến đổi comp thành comp khác bằng việc them data hoặc func

Connect trong redux: sẽ nhận 1 comp => truy cập vào redux store => đưa stage vào comp như là props

withRoute: tim thêm thông tin router và func vào comp, cho phép truy cập hoặc thay đổi route

1. **Controlled Comp >< Uncontrolled Comp**

Controlled comp là comp có các dữ liệu được quản lý bởi state (ví dụ: các input change và update theo state)

Uncontrolled comp là comp có các dữ liệu lấy trực tiếp từ api DOM (ref attribute)

1. **PureComponent**

Là comp có hàm shouldComponentUpdate được override với thuật toán so sánh shallow của react, dùng cho class component

PureComp sẽ không rêrender nếu kết quả so sánh của props và state giống nhau

*Ưu điểm*: giảm số lần rerender

*Nhược điểm*: nếu prop và state phức tạp thì thuật toán shallow sẽ không chạy đúng – vi dụ 1 user lồng nhiều object con và khi update các object con thì oldState.user === newState.user sẽ bằng true => comp không update

1. **Component là gì?**

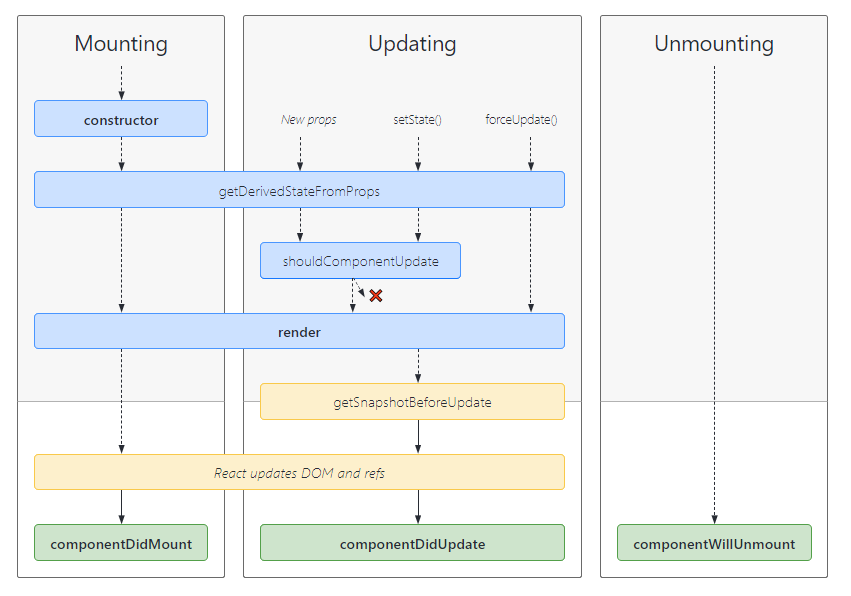
Là những thành phần cho phép chia tách giao diện thành những phần độc lập và có thể tái sử dụng

1. **Class, kế thừa class**

Class là một template dùng để tạo ra đối tượng có thuộc tính và phương thức giống nhau

Class kế thừa là 1 class được kế thừa các thuộc tính và các các action của 1 class khác

1. **Life cycle**



1. **SetState async or sync**

setState để update lại state và sau đó update lại UI nếu cần

setState => getDerivedStateFromProps => shouldComponentUpdate => render => componentDidUpdate

1. **Portal react**

Cho phép tạo ra 1 comp bên ngoài app nhưng vẫn giữ các state và props (popup, notification, …)

1. **Fragment vs <> </>**

Là một common pattern cho phép return nhiều element từ một comp mà không làm sinh ra những element không cần thiếu

< > < />: Không hỗ trợ key và attributes

1. **Spread operator / rest**

Rest: là toán tử cho phép lấy những phần còn lại trong object hoặc array

Spread: clone 1 object hoặc array, nối mảng, kết hợp object

1. **Destructing operator**

Là toán tử cho phép lấy ra chính xác giá trị của thuộc tính trong object hoặc array

1. **Scope block**

Là nới các biến khai báo tồn tại

1. **Closure**

Là hàm con nằm trong một hàm khác

Có quyền truy cập vào phạm vi biến global, biến trong hàm cha, biến local

1. **Arrow function >< normal function**

Là một function đc viết theo cách ngắn ngọn

k rang buộc this

k sử dụng làm function constructor

1. **Callback function**

Là function được truyền vào 1 funtion khác thông qua đối số, và được thực sau khi hàm khác thực thi xong

Ưu điểm:

+ đảm bảo các hàm thực thi đúng trình tự (xử lí các tiến trình bất đồng bộ)

1. **UseMemo, useCallback ( tránh việc re render k cần thiết)**

useCallback: tạo ra 1 memoized callback và chỉ tạo ra callback khi dependencies thay đổi

2 tham số: 1 function, dependencies

useMemo: tạo ra memorized value và chỉ tính toán ra value mới khi dependencies thay đổi

2 tham số: 1 function, dependencies

1. **Optional chanining**

Cho phép đọc giá trị của thuộc tính mà không cần phải kiểm tra thuộc tính đó có hợp lệ hay không

1. **React hook**

Sử dụng hook thay thế cho life cycle trong func comp

1. **Redux**

Là một thư viện tạo ra 1 global state và cho phép component lấy data từ một redux store, dispatch 1 action để update 1 redux-state

*Để truy xuất state*: connect vào store => mapStateToProp truyền vào container => truyền vào component

*Hook:*

* userSelector: dùng để truy xuất state từ redux store.

Khi dispatch 1 action userSelector sẽ so sánh tham chiếu với giá trị trc đó và giá trị hiện tại, nếu khác -> reRender, nếu giống => không rerender

* useDispatch: return về 1 tham chiếu đến dispatch function từ store và đc dùng để dispatch action

1. **React Memo**

**Memoization:** là cách thức tạo bộ nhớ kết quả theo từng bộ input, khi gặp lại bộ input cũ thì không tính toán lại mà trả về kết quả đã lưu

**React Memo:** là 1 HOC, dùng cho Function Comp, tác dụng là chỉ render lại Comp khi props thay đổi với kỹ thuật shallow comparation

1. **Literal string**

Là toán tử cho phép sử dụng mutil line, biến sô, biểu thức … trong string

1. **DOM event / Event listener**

DOM event: là phương thức them sự kiện vào element

Event Listener: là phương thức them sự kiện vào element, dùng xử lý nhìu việc khi 1 event xảy ra, có thể remove event

1. **Custom event**

custom event bằng cách sử dụng new CustomEvent

dispatch event: ele.dispatchEvent(event, option)

event.preventDefault: hủy bỏ các default action

mỗi event có 2 tham số

* Event type: tên của event
* Option:
  + bubbles – hiệu ứng nổi bọt, trigger event từ DOM đăng ký cho tới DOM parents (stop nó: event.stopPropagation()) / capturing trigger event từ parents tới DOM đã đăng ký event
  + cancelable: default action sẽ đc huỷ bỏ

1. **Single Page Application**

Là 1 ứng dụng web giúp cải thiện trải nghiệm người dùng, ứng dụng nằm trong 1 trang duy nhất

* Sau các action trên UI thì không phải load lại toàn bộ page -> Giảm thời gian load page
* Frontend và backend tách biệt, dễ maintain

Hoạt động: Đầu tiên khi tải trang, toàn bộ resource của web js, css, master layout, … sẽ được đc load 1 lần, ở các lần sau, khi chuyển trang ajax sẽ gửi các yêu cầu để lấy dữ liệu cần thiết và update

Nhược điểm: kỹ thuật SEO rất khó để sử dụng

1. **Restful API**

*API application programming interface là các phương thức, giao thức kết nối các thư viện và các ứng dụng khác nhau*

Là các tiêu chuẩn, cách thức thiết kế API cho các ứng dụng web để quản lý resouce và nó hoạt động dựa trên giao thức HTTP

- GET: Trả về một Recourse hay danh sách Recourse.

      - POST: Tạo ra một Recourse mới.

      - PUT: Cập nhật thông tin cho Recourse.

      - DELETE: Xoá Recourse.