=====================**Angular**=======================

Angular interview(<https://www.naukri.com/code360/library/angular-interview-questions>)

How much experience do you have with Angular?

I started to work with Angular from Angular 2. Its from 2017. Angular 2 was released in September 2016. After that Angular team started to release new version in every 6 months. In Angular 5, they introduced httpClient. Angular 8 they introduce Ivy engine to make bundle size small. Ivy engine make it default in Angular 9. Then Angular 15, they introduce standalone component. Angular 17, They make it default. Angular 17, they have introduced Built-in control flow, Deferrable views, Angular Signals.

**Pure pipes** are only executed when the input value changes, while

**impure pipes** are executed on every change detection cycle

**Angular life cycle hook**: A component's **lifecycle** is the sequence of steps that happen between the component's creation and its destruction

1. **Creation**

1.1

1. **Change Detection**
2. **Rendering**
3. **Destructionf**

**Form :** [**https://climbtheladder.com/angular-reactive-forms-interview-questions/**](https://climbtheladder.com/angular-reactive-forms-interview-questions/)

1. **Template Driven:**

* Bind with ngModel
* More code in html
* Less code in ts
* Validation easy

===> FormsModule: exports the required providers and directives for template-driven forms

**ngModel**: This input is actually a control of my form.

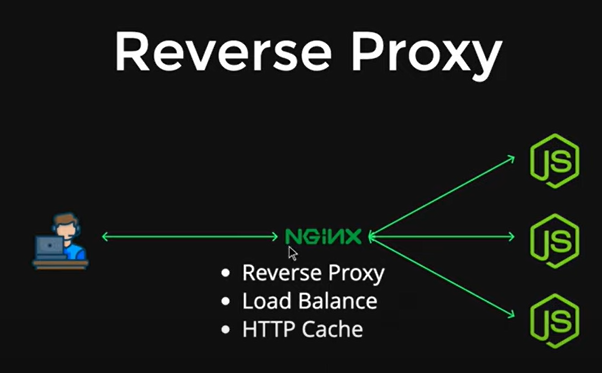
1. **Reactive:** Create programmatically and sync with DOM

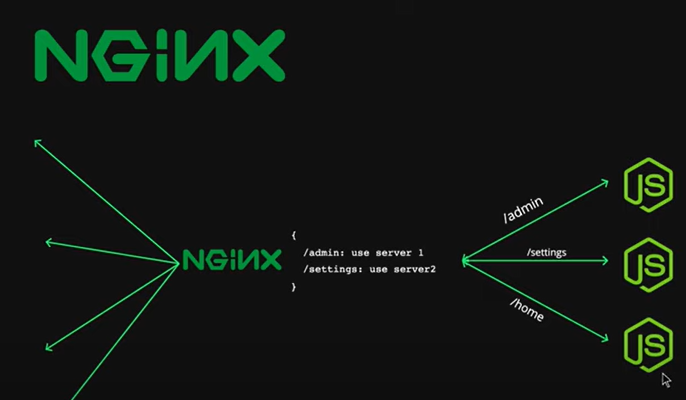
==========================**ngInx**================================

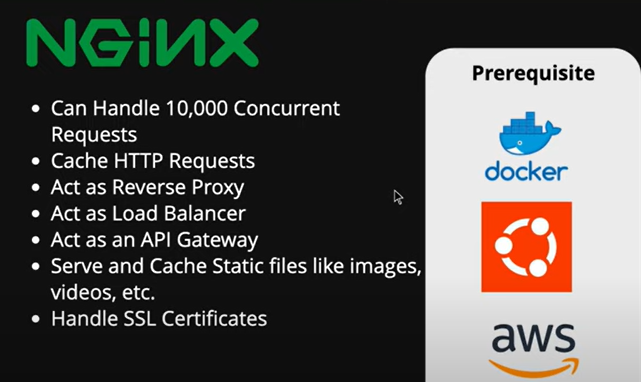
User ========🡺 NhInx ===🡺 server 1

Cline --> request -> server

Server --> response 🡪 client







Nginx is a web server that can also act as a reverse proxy, load balancer, mail proxy, and HTTP cache. It was created by Igor Sysoev in 2002

==============================**Rxjs**==============================

What is RxJS ?

It's an acronym for Reactive Extensions of JavaScript. and it's a JavaScript library for handling Asynchronous data streams and events.

**What is Stream ?**

Values of data over time.

**What is Reactive Programming?**

It's a programming paradigm that deals with asynchronous streams of data.

**What are the advantages of Reactive Programming?**

* Easy to handle streams of data.
* Has many useful operators that can save time: switchMap, concatMap.
* Helps avoid callback hell.
* Makes complex threading easy.
* Cleaner & maintainable code.

**What do you understand by the term Non-Blocking in RxJS?**

It's an algorithm that does not allow threads competing for the same resource to be indefinitely postponed through mutual exclusion of the resource.

**What is Back-Pressure in Reactive Programming?**

It's the resistance(opposing force) on the desired data flow of an application.

**What is an Obserable?**

It's an object that emits values over time.

**What is the difference between an observable and a Promise?**

**Promise:**

* A promise has a single callback at the end of completion or failure.
* A promise is not Lazy. It cannot be cancelled.

**Observable:**

* An observable emits multiple values over time.
* You may cancel an obeservable with an Unsubscribe() method.
* An observable only executes once subscribed to it.
* Provides efficient operators: map, reduce, foreach, filter, reduce, retry, retryWhen.

**What is the difference between Cold and Hot Observables ?**

* Cold observables have their data produced inside the observable.
* Hot observables have their data produced outside the observable.
* Cold Observables only emit data once subscribed to;
* Hot Observables emit values even before the subscription is made. Used when sharing data among many subscribers.

**What are RxJS Operators ?**

* These are methods that can be applied to the observable in order to modify the data stream.

**What is Observers and Subscriptions ?**

* Observers: They are the listeners/ consumers of the data stream; Observers listen /subscribe to the data stream.
* Subscribers: - Objects returned when an observable is subscribed; Useful in canceling the subscription.

**What is Subject ?**

* It's a type of observable where data is multicasted to its observables.

What are different types of Subject ?

* Behaviour Subject & Replay Subject.
* Subject: -Doesn't revive data prior to their subscription.
* Behaviour Subject: - Emits last value to all & new observers.
* Replay Subject: - All observers will revive data prior to their subscription; uses a buffer to hold values and re-emits values on new subscriptions.

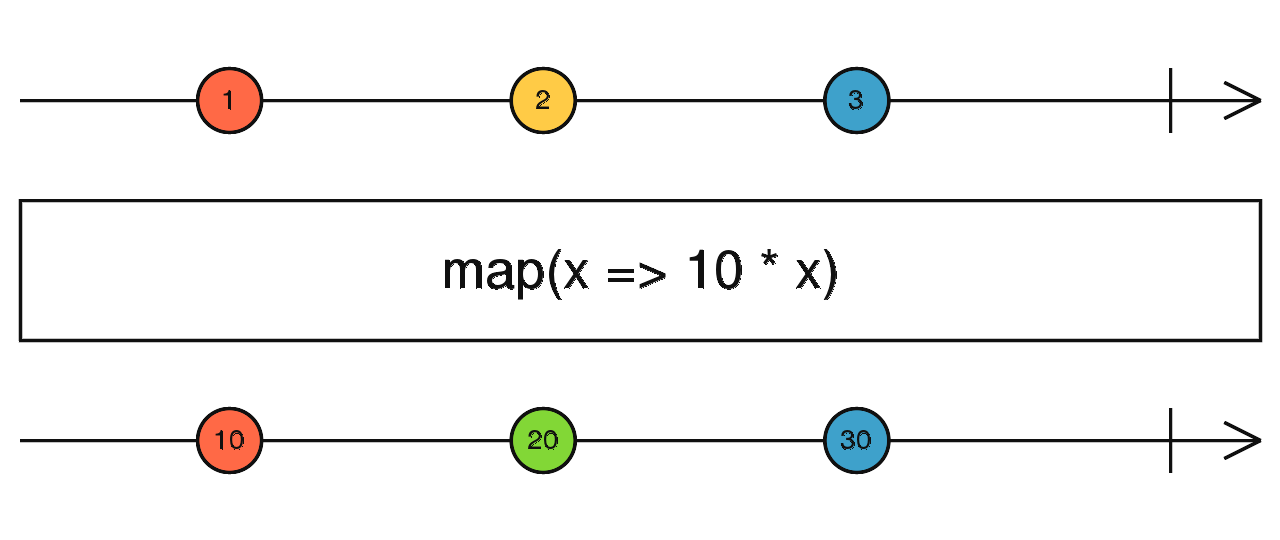
What are Schedulers:

A scheduler controls the execution of when the subscription has to start and be notified.

Types of schedulers: queueScheduler, asapScheduler, asyncScheduler, animationFrameScheduler

What is RxJS Map and What is Higher-Order Observable Mapping?

* RxJS Map: It's a flattening operator used to transform current emitted data values to the desired data format.

[](https://camo.githubusercontent.com/a76b11cd52f618fd2ecbc4735427cb39d572e6b927fe03dbd063e469c8bd350b/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f6d61702e706e67)

const http$ : Observable<Books[]> = this.http.get('/api/books');

http$

.pipe(

tap(() => console.log('HTTP request executed')),

map(res => Object.values(res['payload']))

)

.subscribe(

books => console.log("books", books)

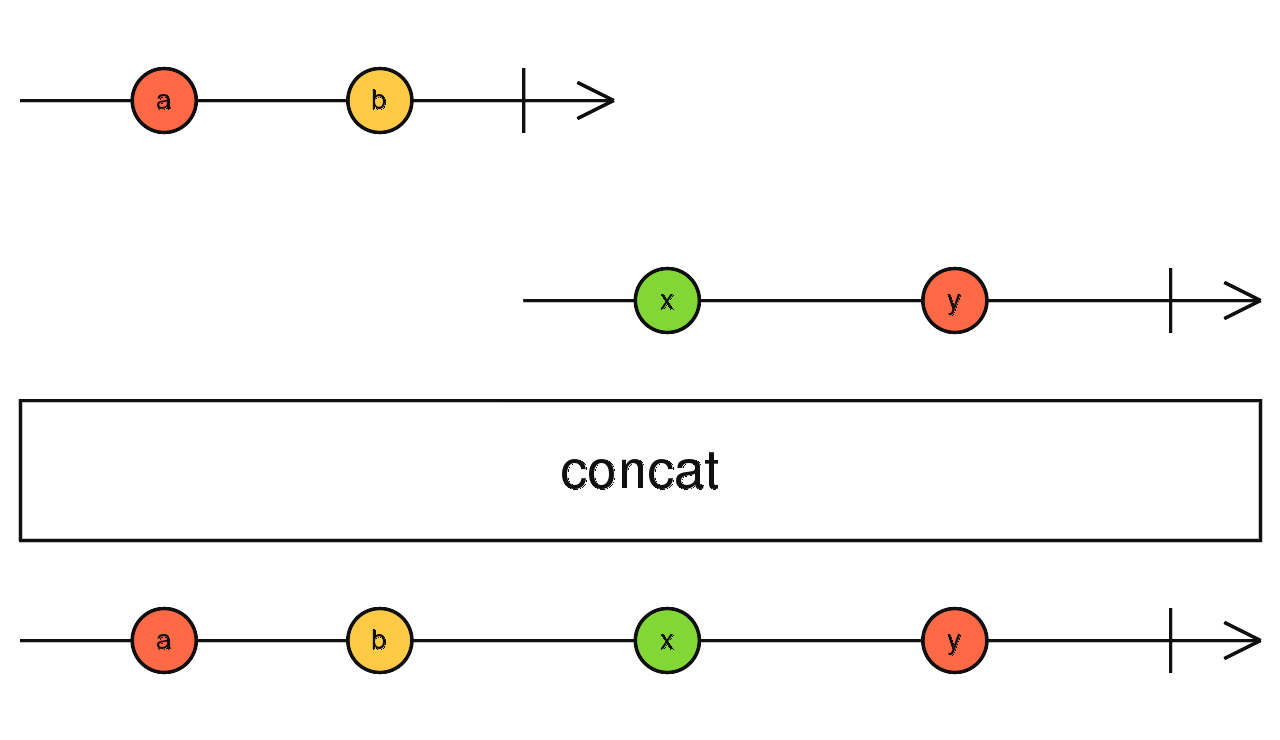
);

* Higher-Order Mapping: These are Rxjs operators used to map source observable values into other observables.

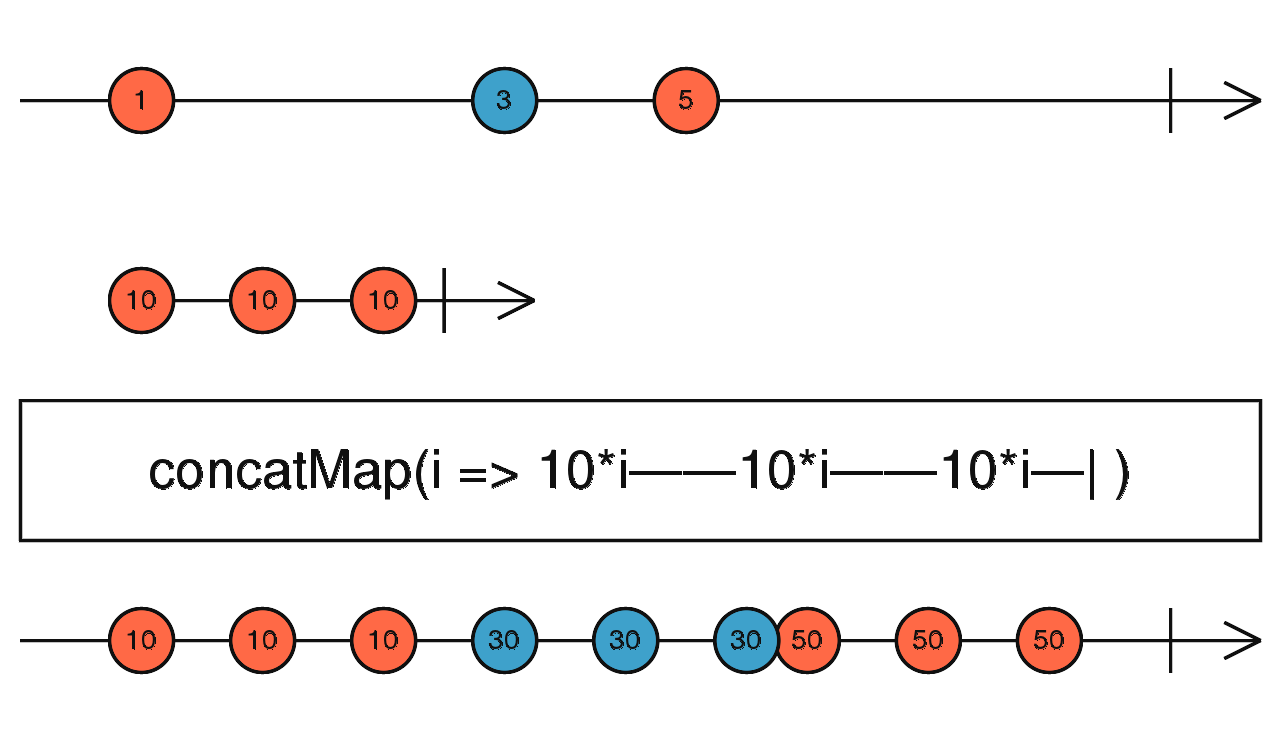
When do we use the switchMap, mergeMap and concatMap?

i) What is RxJS ConcatMap?

* Concat: Joins multiple Observables together, by subscribing to them one at a time and merging their results into the output Observable.
* Used to combine multiple HTTP requests.

[](https://camo.githubusercontent.com/dbd22c4561593da3a9262d6851ba08aa961f219dadceaabc45fa172488c9e29e/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f636f6e6361742e706e67)

* ConcatMap: Projects each source value to an Observable which is merged in the output Observable, in a serialized fashion waiting for each one to complete before merging the next.

[](https://camo.githubusercontent.com/3eacf6ead08001edf8e9af40dc9e845085f5cef9f3d372073b06042c8f7c84f1/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f636f6e6361744d61702e706e67)

Why would you want to use ConcatMap?

* Used to combine multiple HTTP requests; alleviating the need for nested subscribers.
* All HTTP requests are sent to the backend sequentially; Once the previous request has been completed.

this.form.valueChanges

.pipe(

concatMap(formValue => this.http.put("/api/book/",formValue))

)

.subscribe(

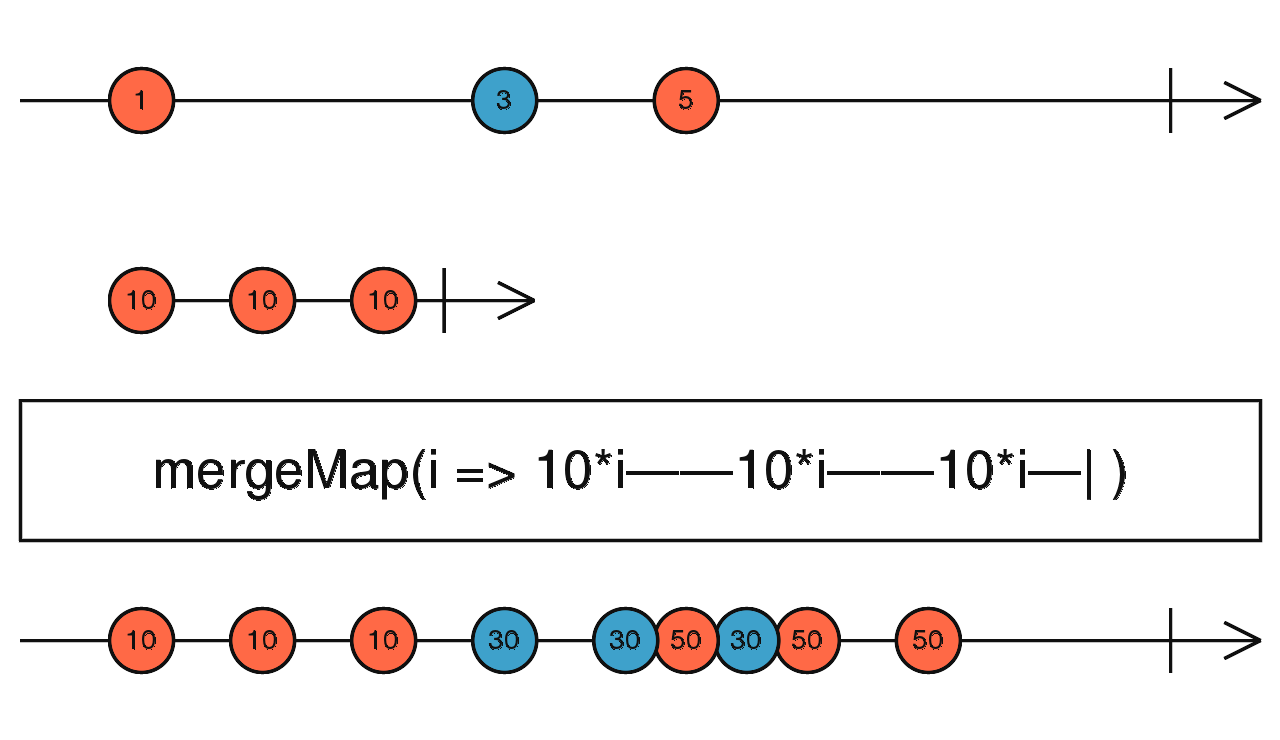
response => ... handle successful ...,

err => ... handle error ...

);

ii) What is RxJS mergeMap?

* MergeMap:Projects each source value to an Observable which is merged in the output Observable.
* Its operation is in parallel, unlike concatMap, it's not sequential in nature.

[](https://camo.githubusercontent.com/a12a1ad90ef849b6ed7bc97027c5d3acee37f0b338d3fdabd8c563891492cb1f/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f6d657267654d61702e706e67)

this.form.valueChanges

.pipe(

mergeMap(formValue =>

this.http.put("/api/book/", formValue))

)

.subscribe(

res => ... handle successful response ...,

err => ... handle error ...

);

iii) What is RxJS switchMap?

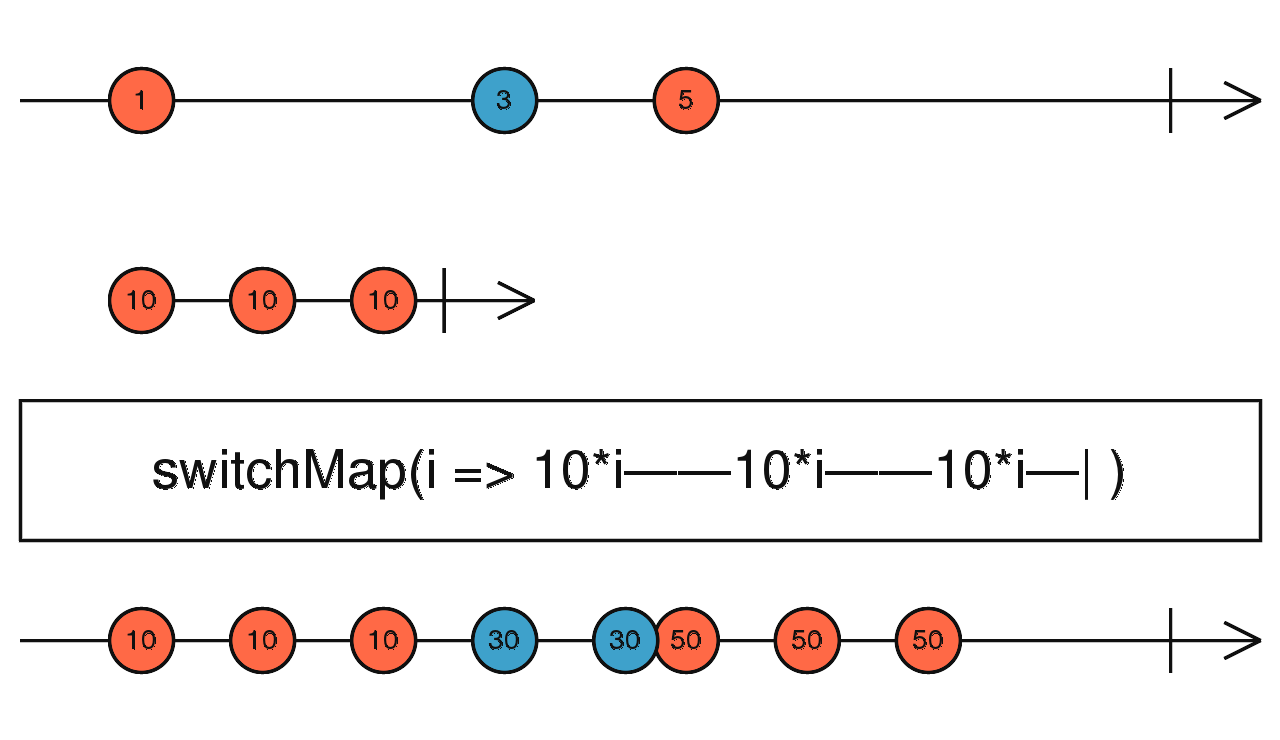
* switchMap: An operator that combines source values into an output observable that represents the most recent projected observable.
* In switching, unlike merging, we'll unsubscribe the previous Observable before subscribing to the new Observable if the new Observable begins to emit the values.

Why would you want to use SwitchMap?

* When you want to get the most rescent updated values the second source anticipated to delay.

UseCase:

a) Update data from local cache or CDN before retrieving from backend.

[](https://camo.githubusercontent.com/645ac9de84185be1a3caaf8974a55df6d523865d2332de55a7ee15fafab88e62/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f7377697463684d61702e706e67)

const searchText$: Observable<string> =

fromEvent<any>(this.input.nativeElement, "keyup")

.pipe(

map(event => event.target.value), // map to form input component value

startWith(""), // avoid spaces

debounceTime(400), // delay request by 400 ms to avoid sending multiple request while user still typing

distinctUntilChanged() // prevent duplicate request on retype

);

const lessons$: Observable<Lesson[]> = searchText$

.pipe(

switchMap(search => (){

const params = new HttpParams().set("search", search);

return this.http.get("/api/book/", {params});

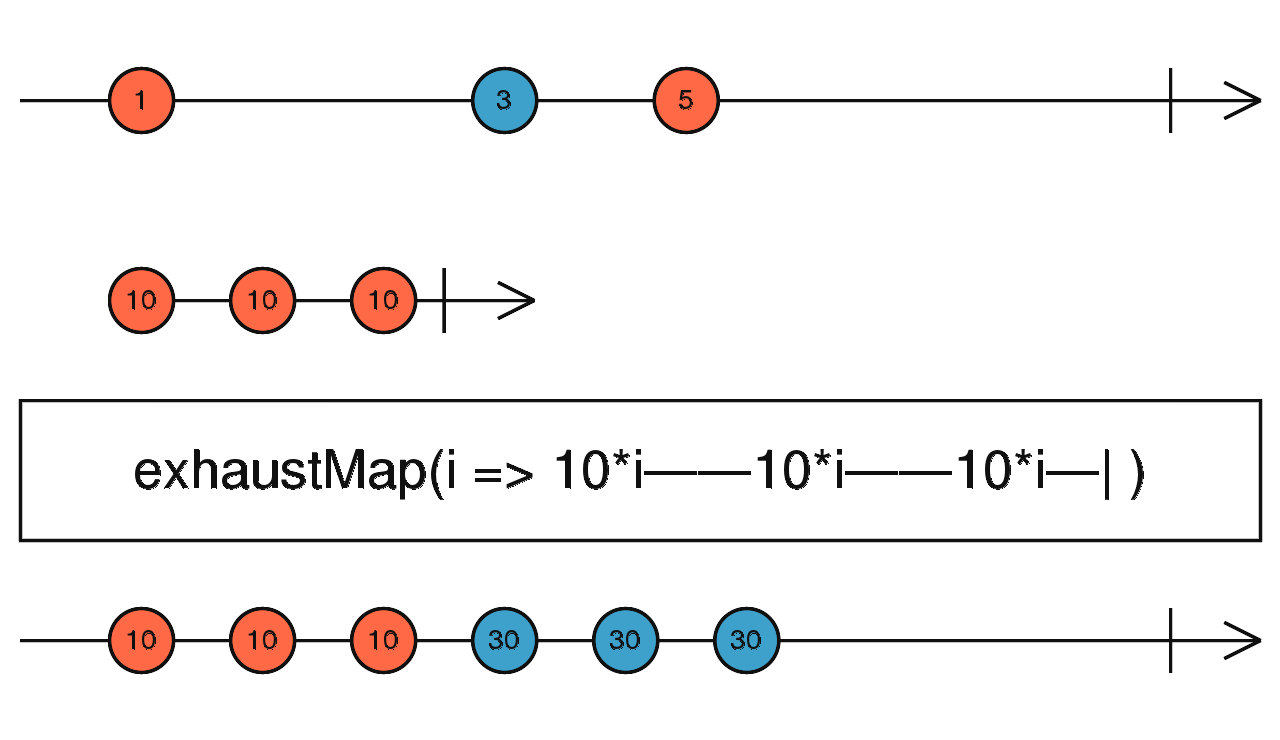
)

)

.subscribe();

iii) What is RxJS exhaustMap?

* exhaust: Converts a higher-order Observable into a first-order Observable by dropping inner Observables while the previous inner Observable has not yet completed.
* It behaves like mergeAll. However, exhaust ignores every new inner Observable if the previous Observable has not yet completed. Once that one completes, it will accept and flatten the next inner Observable and repeat this process.
* ExhaustMap: Projects each source value to an Observable which is merged in the output Observable only if the previous projected Observable has completed.

[](https://camo.githubusercontent.com/b5be100acd8902b63b01171033636737bf32d097af6a2f8edc13381ad025a82c/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f657868617573744d61702e706e67)

const clicks = fromEvent(document, 'click');

const result = clicks.pipe(

exhaustMap(ev => interval(1000).pipe(take(5)))

);

result.subscribe(x => console.log(x));

When we use zip and combineLatest and withLatestFrom ?

* Zip: Combines multiple Observables to create an Observable whose values are calculated from the values, in order, of each of its input Observables.

import { zip, of } from 'rxjs';

import { map } from 'rxjs/operators';

let age$ = of<number>(27, 25, 29);

let name$ = of<string>('Foo', 'Bar', 'Beer');

let isDev$ = of<boolean>(true, true, false);

zip(age$, name$, isDev$).pipe(

map(([age, name, isDev]) => ({ age, name, isDev })),

)

.subscribe(x => console.log(x));

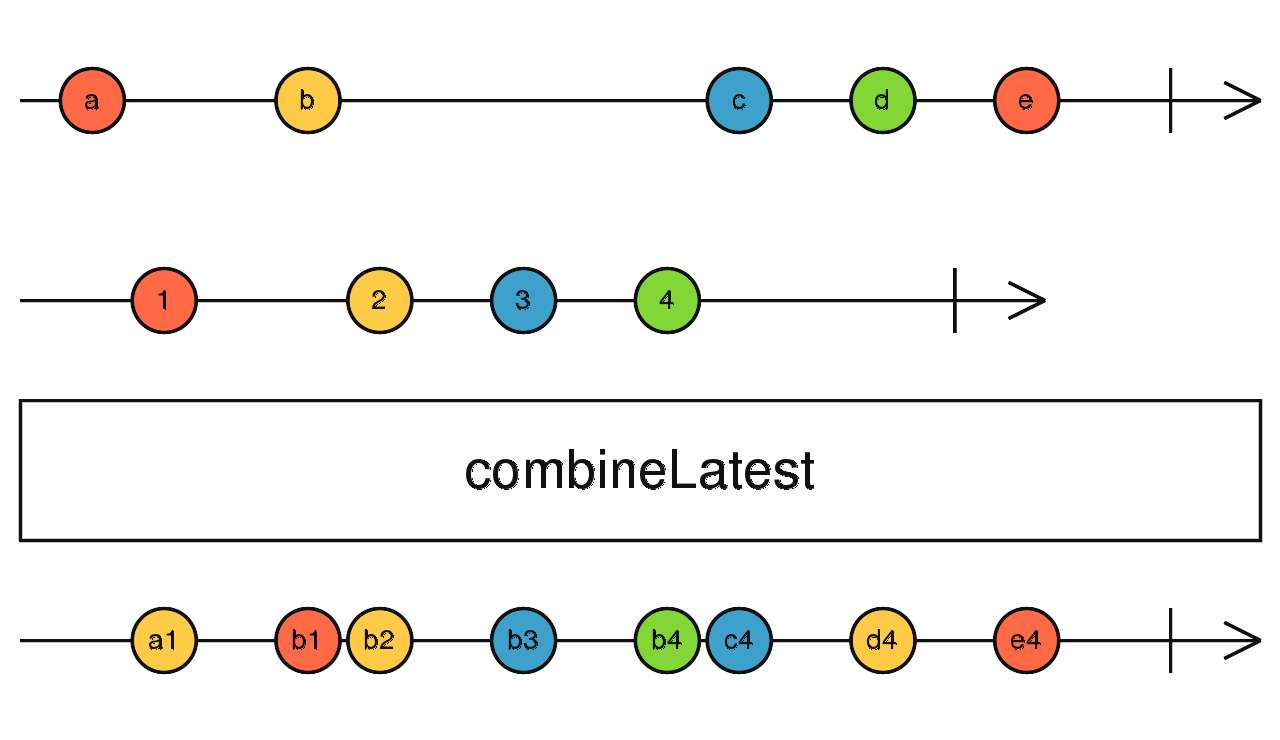
// outputs

// { age: 27, name: 'Foo', isDev: true }

// { age: 25, name: 'Bar', isDev: true }

// { age: 29, name: 'Beer', isDev: false }

* combineLatest: Combines multiple Observables to create an Observable whose values are calculated from the latest values of each of its input Observables.

[](https://camo.githubusercontent.com/f917b5310557388b3413cfa07e673b12eeb577346c7f09703d2e1ae35c7c8b44/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f636f6d62696e654c61746573742e706e67)

import { combineLatest, of } from 'rxjs';

import { delay, starWith } from 'rxjs/operators';

const observables = [1, 5, 10].map(

n => of(n).pipe(

delay(n \* 1000), // emit 0 and then emit n after n seconds

startWith(0),

)

);

const combined = combineLatest(observables);

combined.subscribe(value => console.log(value));

// Logs

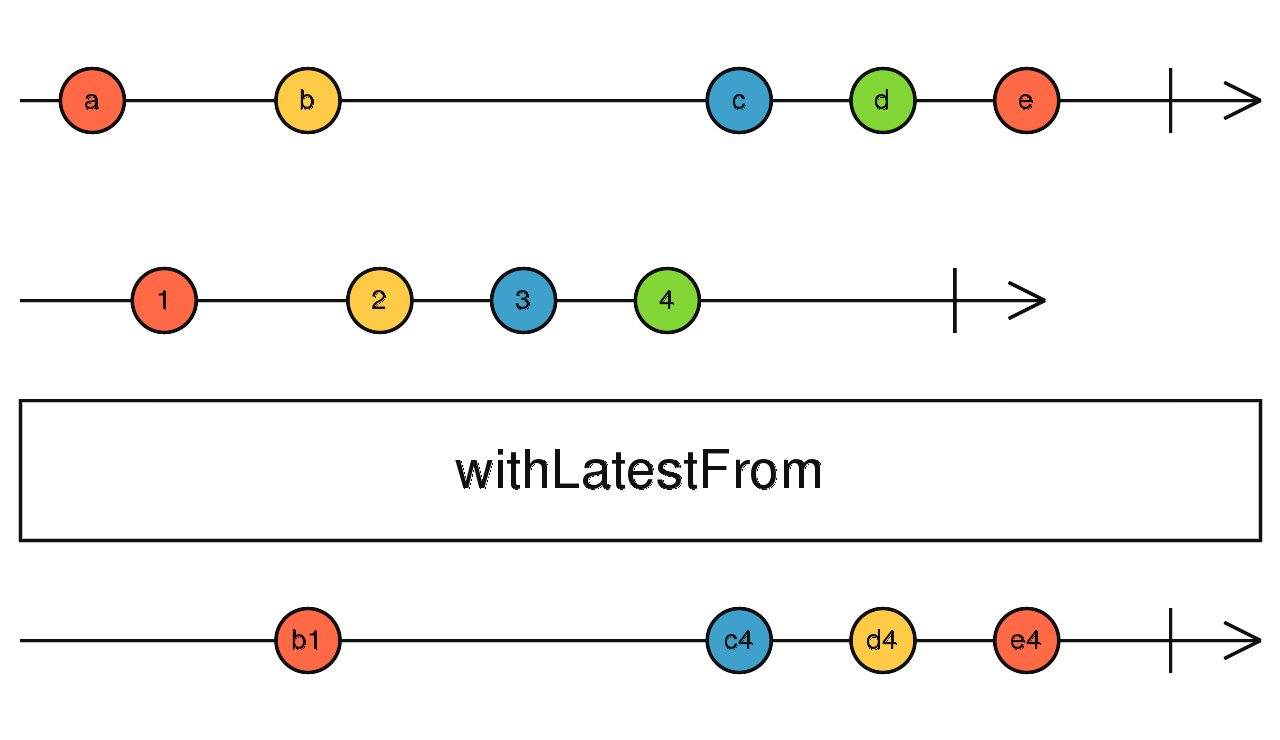
// [0, 0, 0] immediately

// [1, 0, 0] after 1s

// [1, 5, 0] after 5s

// [1, 5, 10] after 10s

* withLatestFrom: Combines the source Observable with other Observables to create an Observable whose values are calculated from the latest values of each, only when the source emits.

[](https://camo.githubusercontent.com/42c0601cde272c30760c5e55ee23c362418e1bc12bb960e8d12e54e19352261c/68747470733a2f2f72786a732d6465762e66697265626173656170702e636f6d2f6173736574732f696d616765732f6d6172626c652d6469616772616d732f776974684c617465737446726f6d2e706e67)

import { fromEvent, interval } from 'rxjs';

import { withLatestFrom } from 'rxjs/operators';

const clicks = fromEvent(document, 'click');

const timer = interval(1000);

const result = clicks.pipe(withLatestFrom(timer));

result.subscribe(x => console.log(x));

============================**end angular**==============================

Redux

**What is Redux?**

Open-Source JavaScript library for managing application state. Used in frontend framework such as React, Angular & Rxjs

What are the core principles of Redux?

-Single source of Truth: All application data is stored in a single object tree.

* State is read only: The state is [immutable](https://en.wikipedia.org/wiki/Immutable_object).
* State can only be change with Pure Functions: These pure functions are called Reduces and they specify how the object tree is should be updated.

**Is there any similarity between Redux and RxJS?**

* Redux uses reactive paradigm to mutate state of the store.
* RxJs uses reactive paradigm to accomplish asynchronous data processing.

What is the difference between Reactive Programming and Imperative Programming?

* Reactive programming is a programming paradime where data is being Pushed to subscribed observers in an application. Where as:
* Imperative programming is a programming paradime where data is being Pulled on request from an application.

**What is the difference between BehaviorSubject and Observable in RxJS?**

* Observable is a consumer/listener while a BehaviourSubject is a synchronous data source that emits data to observers.
* Observables are stateless while BehaviourSubject is stateful.
* Observables create a copy of the data while BehaviourSubject emits data to observers.
* Observables are unidirectional in nature while BehaviorSubject is bi-directional in nature.

=============**Udemy Node js: Course**=================================

1. Html to pug: https://codebeautify.org/html-to-pug-converter

Run time env for js.

REPL (Read User i/p evaluate Print Loop)

Java script:

* Weakly typed( data can be switched)
* OO(data can organized in logical object)
* Versatile(run on browser/directly on server)

Chunk: piece or fragment of data that is received in the stream

Syntax error

Frameork: helper function, tools and rules

**Expree Js:**

**Use:** add new middleware

**Middleware:** middleware functions are JavaScript functions that have access to the request object (req), response object (res), and the next middleware function in the pipeline (next).

**Next: Next is actually a function, a function that will be passed to this function by expressjs (travel to next middle ware) => allow to move next middle ware**

The **app.use ()** function in Express.js **adds middleware to the application’s request-processing** pipeline

**body-parser**extracts the entire body portion of an incoming request stream and exposes it on req.body

**A *template engine***allows us to inject data and then produce HTML.

======================**java script interview questoin**===============================

**Scenario or Code logic question:** https://github.com/lydiahallie/javascript-questions

**Hoisting**: **memory space is set up during the creation phase.**

**Immediately Invoked Function Expressions (IIFE) are JavaScript functions that are executed immediately after they are defined.**

The **CallStack** a mechanism that JavaScript uses to keep track of its execution context.

**De-structuring: Destructuring :** JavaScript expression that allows to unpack of values from arrays

**Stream:** On- going request

**Differences Between Normal Function and Arrow Function**

| **Aspect** | **Arrow functions** | **Normal functions** |
| --- | --- | --- |
| **Anonymous** | Yes | No |
| **Syntax** | Shorter | Longer |
| **Handling multiple expressions** | Difficult | Better |
| **"this" keyword behavior** | Inherits from a broader scope. | Refers to the object it belongs to ("this"). |
| **Constructor functionality** | Not suitable for use as a constructor. | Suitable for use as a constructor. |
| **Arguments object** | Does not have an arguments object | Has an arguments object |
| **Duplicate named parameters** | Not allowed | Allowed |
| **Call, apply, and bind** | Do not change the value of this | Can change the value of this |
| **Lexical scoping** | Lexically scoped | Functionally scoped |

**Temporary dead zone**: Time b/w the declaration and initialization of let and const.

**Describe the state**: where variable are in the scope but they are not declared.

**Map**: method is a higher-order function in JavaScript that creates a new array by performing a specified operation on each element of an existing array.

* **Take call back as argument which execute of every element**.
* **Return new element**

Exp:

**const** numbers = [1, 2, 3, 4, 5];

**const** doubledNumbers = numbers.**map**(**function**(number) {

**return** number \* 2;

});

console.**log**(doubledNumbers);

**Hight Order function:**

* Accept other functions as arguments.
* Return a function as a result.

**Map, filter, reduce**

**The map()**method creates a new [array](https://www.programiz.com/javascript/array) with the results of calling a [function](https://www.programiz.com/javascript/function) for every array element.

* map() does not change the original array.
* map() executes callback once for each array element in order.
* map() does not execute callback for array elements without values.

**Function statement:** The function statement declares a function. **(It can be hoisted)**

* **During hoisting phase** – creation of memory **a(variable)** created a memory and function is assigned to a (variable name)

Exp: **function ab(){}**

**Function expression**: The Javascript Function Expression is used to define a function inside any expression.

* Initially variable defined undefined until the codes hits function line of codel

**Exp:** let *variableName* = function*(x, y)* { *statements...* return *(z)* };

| **Function Declaration** | **Function Expression** |
| --- | --- |
| A function declaration must have a function name. | A function expression is similar to a function declaration without the function name. |
| Function declaration does not require a variable assignment. | Function expressions can be stored in a variable assignment. |
| These are executed before any other code. | Function expressions load and execute only when the program interpreter reaches the line of code. |
| The function in function declaration can be accessed before and after the function definition. | The function in function expression can be accessed only after the function definition. |
| Function declarations are hoisted | Function expressions are not hoisted |
| **Syntax:** function geeksforGeeks(paramA, paramB) { // Set of statements } | **Syntax:** var geeksforGeeks= function(paramA, paramB) { // Set of statements } |

**Anonymous function:** When function is used as a value.

**Named function expression: function** expression with fn name

Parameter: local parameter in this function

**First class function (FOF):** Below characterstics

* **Assigning a function to a variable**

**Exp:** **var a = function (){}**

* **Passing a function as an argument**
* **Returning a function**

**Exp: function a(){**

**Return function b(){**

**Console.log(‘im function b()’}**

**}**

**a()();**

**Arrow Function:** An arrow function is essentially an anonymous function with a shorter syntax

**Callback:** function as an argument (make async call)

**Debouncing** is commonly used to enhance browser performance by ensuring that expensive operations (like **complex calculations, API calls, or DOM updates**) are executed only when necessary.

improve the performance of web applications by limiting the frequency of function calls.

**Debouncing** is a way of delaying the execution of a function until a certain amount of time has passed since the last time it was called.

**Throttling** is a technique that limits how often a function can be called in a given period of time.

========================================end=======================================

=========================**TCS And pWc node js interview question**================

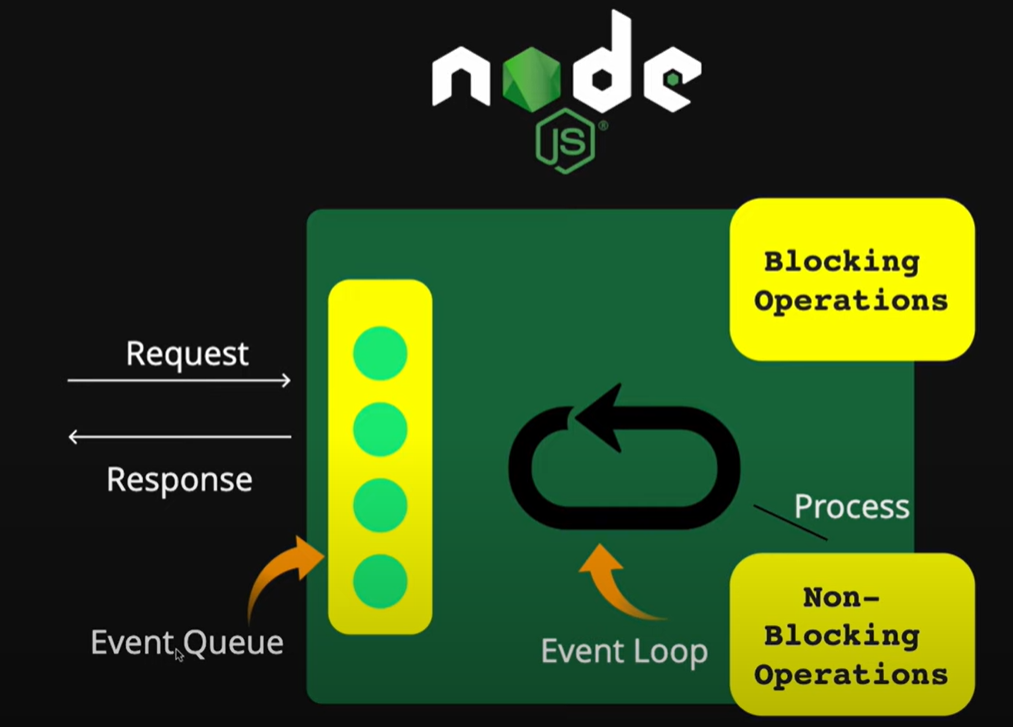
1. Node js server creaiotin code
2. Es6 feature
3. Middleware
4. Why express js and advantage
5. Error handling
6. Callback and overhell come
7. Promise and async
8. How node js works
9. How java script works
10. Call, apply, bind with example
11. Rest api and SOAP Api
12. Put and patch method
13. Routing

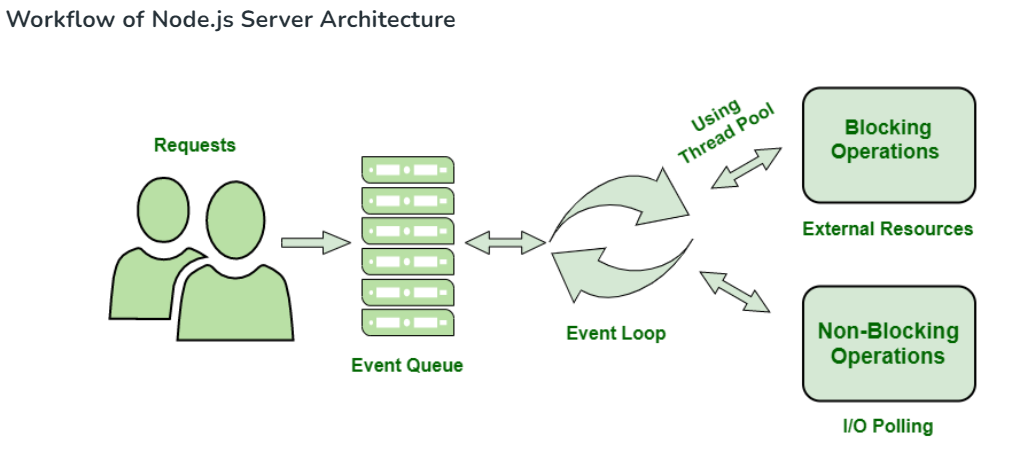
======================**node js interview**=======================

1. Why node js
2. Global

**Node js single thread**: A single-threaded process is the execution of programmed instructions in a single sequence

**event loop** runs one process at a time. (it can only execute one function at a time, and since functions can have multiple instructions, the event loop will execute one instruction at a time.)





* **Requests:**Depending on the actions that a user needs to perform, the requests to the server can be either blocking (complex) or non-blocking (simple).
* **Node.js Server:**The Node.js server accepts user requests, processes them, and returns results to the users.
* **Event Queue:**The main use of Event Queue is to store the incoming client requests and pass them sequentially to the Event Loop.
* **Thread Pool:**The Thread pool in a Node.js server contains the threads that are available for performing operations required to process requests.
* **Event Loop:**Event Loop receives requests from the Event Queue and sends out the responses to the clients.
* **External Resources:**In order to handle blocking client requests, external resources are used. They can be of any type (computation, storage, etc).

===========================CISCO Interiew==========================

what is set

what is week set

what is callback

what is Rxjs

How to remove duplicate array

Get unique value: aabbccddd

combile two object

What is sprd objct

What is clouser with example

What is Angular framework.

What is angular.json

map and forEach diffrence with example

=========Tech mahinda==================

es6 feature

why js use type script

type of variable

folk join in rxjs

Observable and Promise

==============Delta airlines========================

async pipe impure pipe(when change detection occurs) (Handle asynchronous data and automatically unsubscribe the observable and utilize on push chnage detection startegy good for prformance)

1. when getting data from back end server it will take litle bit time

Async task

- It subscribes to an observable or a promise and returns the last emitted value.

- Whenever a new value is emitted, it marks the component to be checked. That means Angular will run Change Detector for that component in the next cycle.

- It unsubscribes from the observable when the component gets destroyed.

sibling communcation (Inout and OutPut)

subject ( observable + Observer)

- Type of observable

- Multi casting (jis stage se subscribe karenge wahi se data mileg) live match

- cross communication can be done

Oservable (Pipr, Subscribe):

- Perform asynchronous operations and handle asynchronous data

- Stream the data in the form of packets

- streans data (Observable) and Subscribe(Observer)

Observer (Next, Error, Complete)

js event and its properties

Entire angular folder structure

angular modules

====================== UHG=====================

1.Angular performance

2.centr of div

3.display property in css

4.viewChild and viewChildren

- ViewChild and ViewChildren are used to communicate between the components to access the data.

- Type of decorator use to access child componnet class into parent componnt.

==> ViewChild: The @ViewChild decorator is used to query a single DOM element from the DOM tree and lets you manipulate it

paramter:

- Selctor: elemnt to query

- Read:

-Static:

@ViewChildren decorator is used to accessing multiple elements. response always be a queryList

\*\* ViewChild:

1. ViewChild and ContentChild are used for component communication in Angular

A ViewChild is any component, directive, or element which is part of a template.(ViewChild decorator allows to query for a single element from the view DOM,)

- Access template of the same compoenent

- accessing the tmplelate child compoennt

- @ViewChild ('selector') variableName:type

Steps:

1- @ViewChild("title"):elem

\*\* ContentChild:

- A ContentChild is any component or element which is projected in the template.(ContentChildren decorator queries a list of elements in the content DOM.)

Just-in-Time (JIT)

JIT compiles your app in the browser at runtime. This was the default until Angular 8.

Ahead-of-Time (AOT)

AOT compiles your app at build time. This is the default since Angular 9.

===================HCL interview=================

1) Center of circle and its centre of screen

2) Chase creation what user give input like 8\*8

3) Life cycle hook

4) Rxjs drawback

5) Real DOM and virtual DOM

6)

========Boing=============

1) Remove white space from string //trim()

2) Remove \_ from string => /\_/g

3) Hoisting

4) Clouser

========================Noida protiviti==================

1) Hoisting with example

2) Clouser with example: A closure in JavaScript is a function that remembers the environment in which it was created, even after that environment no longer exists.

- function foo(outer\_arg) {

function inner(inner\_arg) {

return outer\_arg + inner\_arg;

}

return inner;

}

let get\_func\_inner = foo(5);

console.log(get\_func\_inner(4));

console.log(get\_func\_inner(3));

3) event loop with example

4) Event delegation

5) How browser and send user request to user.

6) Call, Apply and bind with example

7) Diff b/w arrow function and normal function

8) debousing with example

9) React component life cycle

10) UseEffect

11) Life hook in react

12) State Mng - Redux

===========================Aspire solutions=====================

1) userList = [

{"email": "email1@gmail.com"}, {"email": "email2@gmail.com"}, {"email": "email3@gmail.com"},{"email": "email4@gmail.com"}, {"email": "email5@gmail.com"}, {"email": "email6@gmail.com"}

]

// filter email: email1@gmail.com

2) NgRx

3) Rxjs operator

4) forkJoin

5) subject and behaviour

6) Reactive form and template driven

7) Design a reactive form

9) directive

10) DI and services

11) ng-template

12) life cycle hook

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

===============Ericson================

1) scoping in let

2) HOF

3) Call and bind and apply

4) DI and class decorator

==============================MEAN Stack===========================================

### MEAN Interview Questions for 2-3 years Experience

## Node and javascript

1. async,await and promise difference

2. callback hell how to remove this callback hell

3. error first callback

4. Nodejs and Angular difference

5. closure

6. spread operator, rest operator

7. hoisting

8. arrow function and normal function difference

9. middleware in nodejs

10. call, bind, apply with example

11. promise chaining

12. promise.all

13. libuv in nodejs,

14. architecture of nodejs

15. event loop in nodejs

16. destructuring in javascript

17. ES6 features with examples

18. var let const difference

19. null undefined undeclared

20. module.exports and exports

21. diff between typescript and javascript

22. type of Array function.

23. diff between map,filter,forEach

24. cluster,fork, process spawn in nodejs

25. Argument in javascript

26. prototyping (very important)

27. method of object creation, object cloning

28. Authorization and Authentication

29. Difference between slice and splice ( Important )

30. Error Handling in Nodejs

31. What are packages have you used in Nodejs?

32. Difference between package.json and package-lock.json

## Angular

1. component, Directives, Pipe

2. dependency injection

3. routing, wild card routes

4. lifecycle hooks

5. communication between two component if its child parent or independent component

6. rxjs, types of rxjs

7. what is an interceptor and why we use Interceptor.

8. guards ->(CanActivate, CanActivateChild, CanDeactivate,Resolve and CanLoad)

9. knowledge of creating custom directive, pipe, module

10. Observalables and Promise differences with example

11. Difference between Angular 4 and Angular 8

12. What are Decorators in Angular?

13. Difference between Angularjs and Angular 2 {( Important )

14. DataBindings in Angular

15. Subjects vs BehaviorSubjects

16. Lazy loading in Angular

17. Difference between Local Storage, Session Storage and Cookies

18. Starting point of Angular?

## MongoDB

1.Difference between Relational DB and Non-Relational DB

2. What is Aggregation and its pipeline stages?

3. Can we achieve joins in Mongo? Yes, vai $lookup pipeline

4. Sharding

5. Replication