|  |  |
| --- | --- |
| No | Вопросы-ответы |
|  | **HTML** |
| 1 | **What is Critical Rendering Path?**  The Critical Rendering Path is the sequence of steps the browser goes through to convert the HTML, CSS, and JavaScript into pixels on the screen. Optimizing the critical render path improves render performance. |
| 2 | **What is the DOM? How does the DOM work?**  How does DOM works? Document Object Model or DOM is an interface that defines how the browser reads your XML or HTML document. JavaScript is allowed to manipulate structure & style your webpage. Once the browser reads the HTML document, it creates a representational tree known as the Document Object Model. |
| 3 | **Explain the difference between layout, painting and compositing?**  Layout: Browser will determine how much space each element takes up and where to place it.  Painting: This is the process of filling in pixels. It involves drawing out elements.  Compositing: Browser draws element to the screen in the correct order so the page renders correctly. |
| 4 | **What are the Benefits of Server Side Rendering (SSR) Over Client Side Rendering (CSR)?**  SSR позволяет сократить время загрузки страницы, что улучшает пользовательский опыт клиента.  Image result for What are the Benefits of Server Side Rendering (SSR) Over Client Side Rendering (CSR)?  In CSR, the initial page load takes longer compared to SSR because it requires more JavaScript to be downloaded and parsed. A second HTTP request needs to be made to load the content then more JavaScript is needed to generate the template. |
| 5 | **How can I get indexed better by search engines?**  You can get indexed better by search engines by creating sitemaps, auditing them for crawling errors, and submitting them to multiple search engines. Additionally, you should consider optimizing your content for mobile devices and reducing your loading times to speed up crawling and indexing. |
| 6 | **Ways to improve website performance**  Common tips:   1. Choose the Right Hosting Provider 2. Leverage Browser Caching 3. Enable Keep-Alive on Your Web Server 4. Enable GZIP Compression 5. Avoid Landing Page Redirects Whenever Possible 6. Use a Content Delivery Network (CDN) 7. Disable Query Strings for Static Resources 8. Specify a Character Set 9. Minify Your Website’s Scripts 10. Reduce Domain Name System (DNS) Lookups 11. Put CSS at the Top and JavaScript at the Bottom 12. Fix Broken URLs   JS tips:   1. Remove unused JavaScript code 2. Minify Your JavaScript Code 3. Use Gzip compression 4. Keep DOM interaction to a minimum 5. Switch to HTTP/2 6. Delay loading unnecessary JavaScript 7. Use performance improvement tactics that work with other languages   Angular tips:   1. Change Detection Strategy.OnPush 2. Detaching the Change Detector 3. Lazy Loading 4. Ahead-of-Time Compilation (AOT) 5. Serve Scaled Images 6. Minification 7. Enabling Production Mode 8. Optimize Images 9. Code Splitting |
| 7 | **What does async and defer refer in script tag? Describe the difference between <script>, <script async> and <script defer>**  Async - means execute code when it is downloaded and do not block DOM construction during downloading process. Defer - means execute code after it's downloaded, and browser finished DOM construction and rendering process.  The async and defer scripts are executed at different moments. After an async script is downloaded, the browser will pause the document parser, execute the script and resume parsing the document. The defer script, on the other hand, will be executed only when the parser has completed its job.  Usial script blockes the page under it, because it executes synchronously. |
| 8 | **What is desktop first and mobile first design approach?**  When you go for desktop first design, you start a design based on a desktop (a fixed computer, or laptop) computer. Then it is adapted for mobile by means of responsive design. Certain elements are placed below each other, not shown or made narrower / smaller. This way you have the best user experience on desktop.  What is Mobile-First Design? As the term suggests, mobile-first design is an approach in which web designers start product design for mobile devices first. This can be done by sketching or prototyping the web app's design for the smallest screen first and gradually working up to larger screen sizes. |
| 9 | **How to make page responsive?**  To make page responsive, add the following <meta> tag to all your web pages. You may develop a page with flex or grid. |
| 10 | **What are the building blocks of HTML5?**  Semantics — Allowing you to describe more precisely what your content is.  Connectivity — Allowing you to communicate with the server in new and innovative ways.  Offline and storage — Allowing webpages to store data on the client-side locally and operate offline more efficiently.  Multimedia — Making video and audio first-class citizens in the Open Web.  2D/3D graphics and effects — Allowing a much more diverse range of presentation options.  Performance and integration — Providing greater speed optimization and better usage of computer hardware.  Device access — Allowing for the usage of various input and output devices.  Styling — Letting authors write more sophisticated themes. |
| 11 | **What is difference between Select and Datalist?**  Select input element presents options for the users from which they need to select one of them. On the other hand, Datalist presents a list of suggested values to the associated input form (text) field and users are free to select one of those suggested values or type in their own value. |
| 12 | **What are the semantic tags available in html5?**  Article, aside, details, figcaption, figure, footer, header, main, mark, nav, section, summary, time |
| 13 | **Why you would like to use semantic tag?**  One of the most important features of HTML5 is its semantics. Semantic HTML refers to syntax that makes the HTML more comprehensible by better defining the different sections and layout of web pages. It makes web pages more informative and adaptable, allowing browsers and search engines to better interpret content. |
| 14 | **What is accessibility? ARIA role means in a web application**  Accessible Rich Internet Applications ( ARIA ) is a set of roles and attributes that define ways to make web content and web applications (especially those developed with JavaScript) more accessible to people with disabilities.  The definition ARIA role indicates the element is a definition of a term or concept. The dialog role is used to mark up an HTML based application dialog or window that separates content or UI from the rest of the web application or page. |
| 15 | **What is the purpose of the alt attribute on images?**  The required alt attribute specifies an alternate text for an image, if the image cannot be displayed. The alt attribute provides alternative information for an image if a user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader). |
| 16 | **Define semantic markup. What are the semantic meanings  for <section>, <article>, <aside>, <nav>, <header>, <footer> and when/how should each be used in structuring html markup?**  Semantic Markup refers to marking up documents in ways that provide information about the content itself rather than information about the visual styling of the content.  **Section** - represents a generic standalone section of a document, which doesn't have a more specific semantic element to represent it. Sections should always have a heading, with very few exceptions.  **Article -** tag specifies independent, self-contained content. An article should make sense on its own and it should be possible to distribute it independently from the rest of the site.  **Aside -** elementrepresents a portion of a document whose content is only indirectly related to the document's main content. Asides are frequently presented as sidebars or call-out boxes.  **Nav -** element represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents. Common examples of navigation sections are menus, tables of contents, and indexes.  **Header -** element represents a container for introductory content or a set of navigational links.  A <header> element typically contains:   * one or more heading elements (<h1> - <h6>) * logo or icon * authorship information   **Footer -** tag defines a footer for a document or section.  A <footer> element typically contains:   * authorship information * copyright information * contact information * sitemap * back to top links * related documents   You can have several <footer> elements in one document. |
| 17 | **When should you use section, div or article?**  If the content within the element is not semantically related, then use a <div> . If the semantically related content is also able to be self-contained, then use an <article> . Otherwise, use a <section> . |
| 18 | **What is an iframe and how it works?**  An inline frame (iframe) is a HTML element that loads another HTML page within the document. It essentially puts another webpage within the parent page. They are commonly used for advertisements, embedded videos, web analytics and interactive content. |
| 19 | **Explain the use of rel="nofollow", rel="noreferrer", rel="noopener" attribute?**  All three terms (noopener, noreferrer, and nofollow) are values of the rel attribute of the anchor tag in HTML. Arguably, nofollow is the most well-known among web developers and SEO experts.  The **noopener** creates a top-level browsing context that is not an auxiliary browsing context if the hyperlink would create either of those, to begin with (i.e., has an appropriate target attribute value).  The **noreferrer** no Referer header will be included. Additionally, has the same effect as noopener.  The **nofollow** indicates that the current document's original author or publisher does not endorse the referenced document. |
| 20 | **Describe the difference between a cookie, sessionStorage and localStorage?**   | **Local Storage** | **Session Storage** | **Cookies** | | --- | --- | --- | | The storage capacity of local storage is 5MB/10MB | The storage capacity of session storage is 5MB | The storage capacity of Cookies is 4KB | | As it is not session-based, it must be deleted via javascript or manually | It’s session-based and works per window or tab. This means that data is stored only for the duration of a session, i.e., until the browser (or tab) is closed | Cookies expire based on the setting and working per tab and window | | The client  can only read local storage | The client can only read local storage | Both clients and servers can read and write the cookies | | There is no transfer of data to the server | There is no transfer of data to the server | Data transfer to the server is exist | | There are fewer old browsers that support it | There are fewer old browsers that support it | It is supported by all the browser including older browser | |
| 21 | **What does CORS stand for and what issue does it address?**  CORS stands for Cross Orign Resource Sharing and is used to get around the browsers same-origin policy. For security purposes, a browser won't load requests for resources to other domains when those requests are initated by scripts. CORS gets around this issue by supplying a special header that specifies which domains may make XMLHttpRequests for its resources. |
|  | **CSS** |
|  | **What are the possible ways to apply CSS styles to a web page?**  CSS can be applied to HTML or XHTML using three methods: linked, embedded, and inline. |
|  | **What does the cascading portion of CSS mean?**  A CSS file contains the style code for the structure, which includes a heading, listing, paragraph, and links. Now talking about Cascading in CSS means the styling rules. This is the part where CSS can become unnecessarily complicated, even if the ability to use the cascading is occasionally useful. |
|  | **New features in CSS3**  HTML5 video can use CSS and CSS3 to style the video tag. You can change the border, opacity, reflections, gradients, transitions, transformations, and even animations. HTML5 makes adding video super-fast and without having to build a video player. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **What is CSS preprocessor?**  A CSS preprocessor is a program that lets you generate CSS from the preprocessor's own unique syntax. There are many CSS preprocessors to choose from, however most CSS preprocessors will add some features that don't exist in pure CSS, such as mixin, nesting selector, inheritance selector, and so on. |
|  | **What are media queries?**  Media queries are useful when you want to modify your site or app depending on a device's general type (such as print vs. screen) or specific characteristics and parameters (such as screen resolution or browser viewport width). |
|  | **What does box-sizing do?**  The CSS box-sizing property allows us to include the padding and border in an element's total width and height. |
|  | **Explain some pros and cons for CSS animations versus JavaScript animations**  Css animations tend to be a lot smoother, since they can tap into hardware accelerations and various other low level tricks not available directly to you as a JavaScript coder.  Of course, you can trigger and otherwise control these animations from JavaScript, but you won’t be handling every frame in a timer/interval. I have not had a lot of success using what is called “animation” in css, instead I use css transitions, which are much easier to trigger in JavaScript simply by changing a property, the “transform” property being my favorite (unless I am changing a color or opacity). You can set up callbacks for the completion of the animation, and you can even change the property mid-transition and it tends to handle it elegantly/correctly.  There are lots of tricks to making them really smooth, for instance, using transforms rather than changing css properties like top, left, width, height, etc.  The cons are that they might be more limited in what you can do, and they take more experience to figure out the best way to do them. You may have to jump through some hoops if you have a need to, say, stop/pause them halfway in response to some user action. |
|  | **JavaScript Interview Questions** |
|  | Data Types |
|  | What are primitive data types?  String, number, Boolean, null, undefined, symbol, bigint |
|  |  |
|  | **What is undefined property?**  The undefined property indicates that a variable has not been assigned a value, or not declared at all. |
|  | **What is null value?**  In JavaScript, null is a special value that represents an empty or unknown value. For example, let number = null; The code above suggests that the number variable is empty at the moment and may have a value later. |
|  | **What is the difference between null and undefined?**  undefined is a variable that refers to something that doesn't exist, and the variable isn't defined to be anything. null is a variable that is defined but is missing a value. |
|  | **What is the difference between window and document?**  **window**  Each browser tab has its own top-level window object. Each <iframe> (and deprecated <frame>) element has its own window object too, nested within a parent window. Each of these windows gets its own separate global object. window.window always refers to window, but window.parent and window.top might refer to enclosing windows, giving access to other execution contexts. In addition to document and screen described below, window properties include  setTimeout() and setInterval() binding event handlers to a timer  location giving the current URL  history with methods back() and forward() giving the tab's mutable history  navigator describing the browser software  **document**  Each window object has a document object to be rendered. These objects get confused in part because HTML elements are added to the global object when assigned a unique id. E.g., in the HTML snippet  <body>  <p id="holyCow"> This is the first paragraph.</p>  </body>  the paragraph element can be referenced by any of the following:  window.holyCow or window["holyCow"]  document.getElementById("holyCow")  document.querySelector("#holyCow")  document.body.firstChild  document.body.children[0] |
|  | **What is isNaN?**  Definition and Usage. In JavaScript NaN is short for "Not-a-Number". The isNaN() method returns true if a value is NaN. The isNaN() method converts the value to a number before testing it. |
|  | What is the difference between let, const and var?   |  |  |  | | --- | --- | --- | | **var** | **let** | **const** | | The scope of a *var*variable is functional scope. | The scope of a*let* variable is block scope. | The scope of a *const* variable is block scope. | | It can be updated and re-declared into the scope. | It can be updated but cannot be re-declared into the scope. | It cannot be updated or re-declared into the scope. | | It can be declared without initialization. | It can be declared without initialization. | It cannot be declared without initialization. | | It can be accessed without initialization as its default value is “undefined”. | It can be accessed without initialization as its default value is “undefined”. | It cannot be accessed without initialization, as it cannot be declared without initialization. | | hoisting done , with initializing as ‘default’ value | Hoisting is done , but not initialized (this is the reason for error when we access the let variable before declaration/initialization | Hoisting is done, but not initialized (this is the reason for error when we access the const variable before declaration/initialization |   **Note:**Sometimes, users face the problem while working with the *var*variable as they change the value of it in the particular block. So, users should use the *let* and *const* keyword to declare a variable in JavaScript. |
|  | **What are the differences between undeclared and undefined variables?**  Undefined: It occurs when a variable has been declared but has not been assigned with any value. Undefined is not a keyword. Undeclared: It occurs when we try to access any variable that is not initialized or declared earlier using var or const keyword. |
|  | **What are global variables?**  A global variable is a variable that is declared in the global scope in other words, a variable that is visible from all other scopes. In JavaScript it is a property of the global object. |
|  |  |
|  | **What is NaN property?**  In JavaScript, NaN stands for Not a Number. It represents a value which is not a valid number. It can be used to check whether a number entered is a valid number or not a number. |
|  | **What are classes in ES6?**  There are two types of Class in ES6: parent class/super class: The class extended to create new class are know as a parent class or super class. child/sub classes: The class are newly created are known as child or sub class. Sub class inherit all the properties from parent class except constructor. |
|  |  |
|  |  |
|  | **How do you copy properties from one object to other?**  Object.assign(), JSON.stringify – JSON.parse, spread operator |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **What is Hoisting?**  JavaScript Hoisting refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code. Hoisting allows functions to be safely used in code before they are declared. |
|  | **How do you assign default values to variables?**   1. The OR (||) Operator 2. The OR Assignment (||=) Operator 3. The Nullish Coalescing Operator (??) 4. The Nullish Assignment Operator (??=) 5. Function Default Values 6. Destructuring Assignment With Default Values |
|  | Context, Scope proto and Prototype |
|  | **What is the difference between Call, Apply and Bind?**  **call :** binds the this value, invokes the function, and allows you to pass a list of arguments.  **apply :** binds the this value, invokes the function, and allows you to pass arguments as an array.  **bind :** binds the this value, returns a new function, and allows you to pass in a list of arguments. |
|  | **What is scope in javascript?**  Scope in JavaScript refers to the current context of code, which determines the accessibility of variables to JavaScript. The two types of scope are local and global: Global variables are those declared outside of a block. Local variables are those declared inside of a block. |
|  |  |
|  | **What is prototype chain?**  Each object has a private property which holds a link to another object called its prototype. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. By definition, null has no prototype, and acts as the final link in this prototype chain. |
|  | JSON |
|  | **What is JSON and its common operations?**  JSON.parse()  JSON.stringify() |
|  | **How do you parse JSON string?**  Use the JavaScript function JSON.parse() to convert text into a JavaScript object: const obj = JSON.parse('{"name":"John", "age":30, "city":"New York"}'); Make sure the text is in JSON format, or else you will get a syntax error. |
|  |  |
|  | **What is the purpose JSON stringify?**  The JSON.stringify() method converts a JavaScript object or value to a JSON string, optionally replacing values if a replacer function is specified or optionally including only the specified properties if a replacer array is specified. |
|  |  |
|  | Array methods |
|  |  |
|  |  |
|  | **What array methods do you know?**  1. forEach()  2. includes()  3. filter()  4. map()  5. reduce()  6. some()  7. every()  8. sort()  9. Array.from()  10. Array.of() |
|  | **What is the difference between Array.forEach() and Array.map()?**  The forEach() method does not create a new array based on the given array. The map() method creates an entirely new array. The forEach() method returns “undefined“. The map() method returns the newly created array according to the provided callback function. |
|  | Functions |
|  | **What are lambda or arrow functions?**  These are anonymous functions with their own special syntax that accept a fixed number of arguments, and operate in the context of their enclosing scope - ie the function or other code where they are defined. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **What is a pure function?**  A Pure Function is a function (a block of code) that always returns the same result if the same arguments are passed. It does not depend on any state or data change during a program's execution. Rather, it only depends on its input arguments. |
|  | **What are closures?**  In JavaScript, a closure is a function that references variables in the outer scope from its inner scope. The closure preserves the outer scope inside its inner scope. To understand the closures, you need to know how the lexical scoping works first. |
|  | **What is IIFE(Immediately Invoked Function Expression)?**  An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. |
|  | **What is a callback function?**  A JavaScript callback is a function which is to be executed after another function has finished execution. A more formal definition would be - Any function that is passed as an argument to another function so that it can be executed in that other function is called as a callback function. |
|  |  |
|  | **Async JavaScript** |
|  | **What is a promise?**  A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future. |
|  | **Why do you need a promise?**  Promises are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code. |
|  | **What are the three states of promise?**  A Promise is in one of these states: pending: initial state, neither fulfilled nor rejected. fulfilled: meaning that the operation was completed successfully. rejected: meaning that the operation failed. |
|  | **Why do we need callbacks?**  Need of Callback Functions. We need callback functions because many JavaScript actions are asynchronous, which means they don't really stop the program (or a function) from running until they're completed, as you're probably used to. Instead, it will execute in the background while the rest of the code runs. |
|  | **What is a callback hell?**  Callback hell in Node. js is the situation in which we have complex nested callbacks. In this, each callback takes arguments that have been obtained as a result of previous callbacks. This kind of callback structure leads to lesser code readability and maintainability. |
|  | **What is promise chaining?**  Method Chaining is a programming strategy that simplifies and embellishes your code. It is a mechanism of calling a method on another method of the same object. this keyword in JavaScript refers to the current object in which it is called. |
|  |  |
|  |  |
|  | **What is the use of setTimeout?**  The setTimeout function is a native JavaScript function. It sets a timer (a countdown set in milliseconds) for an execution of a callback function, calling the function upon completion of the timer. |
|  | **What is the use of setInterval?**  The setInterval() function is commonly used to set a delay for functions that are executed again and again, such as animations. You can cancel the interval using clearInterval() . If you wish to have your function called once after the specified delay, use setTimeout() . |
|  | **What is an event loop?**  JavaScript has a runtime model based on an event loop, which is responsible for executing the code, collecting and processing events, and executing queued sub-tasks. |
| 12. | **What is call stack?**  A call stack is a mechanism for an interpreter (like the JavaScript interpreter in a web browser) to keep track of its place in a script that calls multiple functions — what function is currently being run and what functions are called from within that function, etc. |
|  | **Common** |
|  |  |
|  |  |
|  | **How do you validate an email in javascript?**   * Uppercase and lowercase letters (A-Z and a-z) * Numeric characters (0-9) * Special characters - ! # $ % & ' \* + - / = ? ^ \_ ` { | } ~ * Period, dot, or full stop (.) with the condition that it cannot be the first or last letter of the email and cannot repeat one after another. |
|  |  |
|  |  |
|  |  |
|  | **What are modules?**  A module in JavaScript is just a file containing related code. In JavaScript, we use the import and export keywords to share and receive functionalities respectively across different modules. The export keyword is used to make a variable, function, class or object accessible to other modules. |
|  | **Why do you need modules?**  You create modules to better organize and structure your codebase. You can use them to break down large programs into smaller, more manageable, and more independent chunks of code which carry out a single or a couple of related tasks. |
|  |  |
|  |  |
|  | **What is a rest parameter?**  The rest parameter syntax allows a function to accept an indefinite number of arguments as an array, providing a way to represent variadic functions in JavaScript. |
|  | **What is a spread operator?**  The JavaScript spread operator ( ... ) allows us to quickly copy all or part of an existing array or object into another array or object. |
|  |  |
|  |  |
|  | **What is nodejs?**  As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. |
|  | **Core Angular** |
|  |  |
|  |  |
|  |  |
|  | **What is the difference between component and directive?**  Component is used to break up the application into smaller components. But Directive is used to design re-usable components, which is more behavior-oriented. That is why components are widely used in later versions of Angular to make things easy and build a total component-based model. |
|  | **What are types of directives?**  The three types of directives in Angular are attribute directives, structural directives, and components. |
|  | **What are pipes?**  Pipes are simple functions to use in template expressions to accept an input value and return a transformed value. Pipes are useful because you can use them throughout your application, while only declaring each pipe once. |
|  |  |
|  |  |
|  | **One way vs two way bindings**  One-way data binding will bind the data from the component to the view (DOM) or from view to the component. One-way data binding is unidirectional. You can only bind the data from component to the view or from view to the component.  Two-way data binding in Angular will help users to exchange data from the component to view and from view to the component. It will help users to establish communication bi-directionally.  Two-way data binding can be achieved using a ngModel directive in Angular. The below syntax shows the data binding using (ngModel), which is basically the combination of both the square brackets of property binding and parentheses of the event binding. |
|  | **What are component lifecycle?**  A component has a lifecycle managed by Angular. Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM. |
|  | **What are lifecycle hooks?**  Here is the complete lifecycle hook interface inventory:   * ngOnChanges - called when an input binding value changes * ngOnInit - after the first ngOnChanges * ngDoCheck - after every run of change detection * ngAfterContentInit - after component content initialized * ngAfterContentChecked - after every check of component content * ngAfterViewInit - after component's view(s) are initialized * ngAfterViewChecked - after every check of a component's view(s) * ngOnDestroy - just before the component is destroyed |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **What types of forms presented in Angular?**  Angular provides two different approaches to handling user input through forms: reactive and template-driven. Both capture user input events from the view, validate the user input, create a form model and data model to update, and provide a way to track changes. |
|  | **What is the difference between Reactive and Template driven forms?**  Template Driven Forms are based only on template directives, while Reactive forms are defined programmatically at the level of the component class. Reactive Forms are a better default choice for new applications, as they are more powerful and easier to use. |
|  |  |
|  | **What is Control Value Accessor?**  A ControlValueAccessor acts as a bridge between the Angular forms API and a native element in the DOM. Any component or directive can be turned into ControlValueAccessor by implementing the ControlValueAccessor interface and registering itself as an NG\_VALUE\_ACCESSOR provider. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **ng-template, ng-container, ng-content** |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Angular Features |
|  | **Input and Output decorators**  @Input() and @Output() give a child component a way to communicate with its parent component. @Input() lets a parent component update data in the child component. Conversely, @Output() lets the child send data to a parent component. |
|  | **Dependency Injection** |
|  |  |
|  |  |
|  | **How Angular understands what dependency to use?**  Dependency injection, or DI, is one of the fundamental concepts in Angular. DI is wired into the Angular framework and allows classes with Angular decorators, such as Components, Directives, Pipes, and Injectables, to configure dependencies that they need.  Two main roles exist in the DI system: dependency consumer and dependency provider.  Angular facilitates the interaction between dependency consumers and dependency providers using an abstraction called Injector. When a dependency is requested, the injector checks its registry to see if there is an instance already available there. If not, a new instance is created and stored in the registry. Angular creates an application-wide injector (also known as "root" injector) during the application bootstrap process, as well as any other injectors as needed. In most cases you don't need to manually create injectors, but you should know that there is a layer that connects providers and consumers. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **What is the difference between ViewChild/children and ContentChild/children**  ViewChildren is different from the ViewChild . ViewChild always returns the reference to a single element. If there are multiple elements the ViewChild returns the first matching element, ViewChildren always returns all the elements as a QueryList. |
|  | **Change Detection** |
|  | **View Tree in Angular?**   * Display hierarchical data in a tree-view structure. * Load a wide range of nodes with optimal performance. * Drag and drop multiple selected tree nodes anywhere. * Select multiple nodes using built-in check boxes. * Edit node text in-line with editable nodes support. * Easily customize nodes, and expand or collapse icons. |
|  | **CD Stategies?**  Angular Change Detection Strategy are the methods by which the updates to the component is tracked and component is triggered to Re-render. There are majorly 2 Change Detection Strategy in Angular. We can configure the Change Detection Strategy for the Component inside the Decorator. Default Strategy. onPush Strategy. |
|  | **How to create custom strategy?**  To create a custom preloading strategy, Angular provides PreloadingStrategy class with a preload method. We need to create a service by implementing PreloadingStrategy and overriding its preload method. To enable custom preloading strategies, we need to configure our preloading strategy service with RouterModule. |
|  | **How to trigger cd manually?**  We may trigger event by cdr.markForCheck and cdr.detectChanges. |
|  | **Difference between markForCheck, detectChanges, tick?**  1- The change detector is detached from the view ( see detach )  2- An update has happened but it hasn't been inside the Angular Zone, therefore, Angular doesn't know about it. |
|  | **Detach vs Reattach?** |
|  | **What actions triggers change detection?**  We can trigger change detection manually by using detectChanges() , ApplicationRef. tick() , and markForCheck() that we mentioned earlier on AsyncPipe . detectChanges() on ChangeDetectorRef which will run change detection on this view and its children. |
|  | **What events are triggering CD with OnPush strategy?**  OnPush strategy is known to trigger ChangeDetection in the following cases :   * when a DOM event the component listens to was received * when the |async pipe receives a new event * when an @Input() * when ChangeDetectorRef::markForCheck is explicitly called (or any other similar methods like, ApplicationRef::tick, ChangeDetectorRef::detectChanges   So when a DOM event is triggered on a component, it won't affect CD on a sibling component. BUT it will be triggered on every parent component in his hierarchy wether Default or OnPush. This does not happend when calling detectChanges for example where only the component gets CD. |
|  | **What is the difference between Observable and Promise?**  The biggest difference is that Promises won't change their value once they have been fulfilled. They can only emit (reject, resolve) a single value. On the other hand, observables can emit multiple results. The subscriber will be receiving results until the observer is completed or unsubscribed from. |
|  | **Different types of Subjects?**  There are four types of Subjects available based on how they behave:  Subject – No initial value or replay available.  BehaviouralSubject – requires an initial value and emits current values to new subscribers.  AsyncSubject – Emits latest values to subscribers on completion of the async task.  ReplaySubject - replays old values to new subscribers when they first subscribe. |
|  | **Difference between Observable and Subject?**  The one and major difference between observable and the subject is that observables are unicast (each subscribed Observer owns an independent execution of the Observable) while the subject is multicast which means it is like an event emitter, it maintains the registry of listener(s) and updates each listener every time the subject changes. |
|  | **BehaviorSubject vs AsyncSubject vs ReplaySubject vs Subject?**  While the BehaviorSubject and ReplaySubject both store values, the AsyncSubject works a bit different. The AsyncSubject is aSubject variant where only the last value of the Observable execution is sent to its subscribers, and only when the execution completes. |
|  | **What are hot, cold and warm streams?**  An Observable is cold when data is produced inside the Observable and the Observable is hot when the data is produced outside the Observable. As we just saw the hot Observable is able to share data between multiple subscribers. We call this behaviour “multicasting”. |
|  | **How to unsubscribe?**   * 1. Use the unsubscribe method   2. 2. Use Async | Pipe   3. Use RxJS take\* operators:   + takeUntil(notifier)   + takeWhile(predicate)   1. Use RxJS first operator   2. Use Decorator to automate Unsubscription   3. Use tslint (Some might need a reminder by tslint, to remind us in the console that our components or directives should have a ngOnDestroy method when it detects none.) |
|  | **Different unsubscribe approaches, should know pros and const of each.**  Specifically, we must unsubscribe before Angular destroys the component. Failure to do so could create a memory leak. We unsubscribe from our Observable in the ngOnDestroy method. |
|  | **What are http interceptors?**  HTTP Interceptors is a special type of angular service that we can implement. It's used to apply custom logic to the central point between the client-side and server-side outgoing/incoming HTTP request and response. Keep in mind that the interceptor wants only HTTP requests. |
|  |  |
|  | **Angular Routing** |
|  | **What is the purpose of routing in SPA?**  Routing lets you display specific views of your application depending on the URL path. To add this functionality to your sample application, you need to update the app. module. |
|  | **How SPA navigation different from MPA navigation?**  The difference between SPA and MPA is that MPAs secures each page to its core. Therefore, it takes more time and effort to maintain the security; hence, page loading time is more. SPAs secure endpoints faster, but the security level is low. SPAs generally rely on JavaScript. |
|  | **How to create a route?**   * Import RouterModule and Routes into your routing module. ... * Define your routes in your Routes array. ... * Add your routes to your application. |
|  |  |
|  |  |
|  |  |
|  | **What is the purpose of ‘pathMatch: full’ property?**  pathMatch = 'prefix' tells the router to match the redirect route when the remaining URL begins with the redirect route's prefix path. pathMatch: 'full' means, that the whole URL path needs to match and is consumed by the route matching algorithm. |
|  | **How to pass data through router?**   * Using routerLink directive. app.component.html. * Using navigateByUrl method. app.component.ts. * Using getCurrentNavigation method. app.component.ts. * Using history.state in ngOnInit. app.component.ts. * Using getState in Location service. app.component.ts. |
|  |  |
|  |  |
|  |  |
|  | **Name all guards?**  There are 5 types of guards in Angular namely CanActivate, CanActivateChild, CanDeactivate, Resolve and CanLoad. |
|  | **What are the purpose of each guard?**  **CanActivate** - route guard preventing access to the specific route  **CanActivateChild** - CanActivateChild is almost similar to CanActivate interface, the only difference is CanActivate is used to control the accessibility of the current route but CanActivateChild is used to prevent access to child routes of a given route, so by using this you don’t need to add canActive on each child route, in other words, you just need to add canActiveChild to parent route and it will work for child routes as well.  **CanDeactivate** - This route guard is used to keep the user from navigating away from a specific route. This guard can be useful when you want to prevent a user from accidentally navigating away without saving or some other undone tasks.  This route guard is a little bit different in implementation from the above-mentioned routes as it involves defining a method in the component class itself, which gets called whenever the user tries to navigate away from the route.  **Resolve** - Complex angular applications involve data communication between components, sometimes data is so heavy that it is not possible to pass data through query params. To handle this situation angular has provided Resolve Guard.  Now what the Resolve guard does is resolving data based on implemented code and pass that data to the component.  **CanLoad** - Modules in angular can be loaded all at once or be lazy loaded . By default angular load all modules eagerly. To implement lazy loading we use loadChildren in route definition. The main advantage of lazy loading is that it reduces the loading time of application by downloading only the required modules.  Now here if we want to prevent navigation of an unauthorized user we can use CanActivate Guard, that will do the job but also download the module. Now to control the navigation as well as prevent downloading of that module we can use CanLoad Guard. |
|  |  |
|  | **What is Lazy loading?**  Lazy loading is the process of loading components, modules, or other assets of a website as they're required. Since Angular creates a SPA (Single Page Application), all of its components are loaded at once. This means that a lot of unnecessary libraries or modules might be loaded as well. |
|  | **How to implement lazy loading?**  Create a new Angular project.  Create a module and separate routing file named lazy-loading.  Create a component named lazy-demo within the lazy-loading module.  Adding a link in the header on whose route we will implement lazy loading. |
|  |  |
|  |  |
|  | **What are chunks?**  chunks value can be used to filter modules between sync and async chunks. Its value can be initial , async or all . initial means only add files to the chunk if they are imported inside sync chunks. async means only add files to the chunk if they are imported inside async chunks( async by default) |
|  |  |
|  | **Optimization** |
|  | **Optimization techniques within Angular?**   * OnPush change detection strategy * Detach Change Detection/NgZone * Using Pure Pipes * AOT Compilation * Lazy Loading * Trackby * Web Workers * Preloading Modules * Resolve Guards * Unsubscribe from Observables |
|  |  |
|  | **What is trackBy?**  The trackBy function takes the index and the current item as arguments and needs to return the unique identifier for this item. Now when you change the collection, Angular can track which items have been added or removed according to the unique identifier and create or destroy only the items that changed. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |