|  |
| --- |
| HR |
| Introduce yourself? Name, age, education, job history, current technical stack. |
| What’s your salary expectations? Search about salary ranges in the job country. |
| What do you know about our company?  Company’s products and vision) > website. |
| Please tell us about the situation, when you showed your leadership? |
| Please tell about the situation, when you had a conflict and how you resolved it? Listen for everyone involved, friendly discussions and motivation. |
| Why do you want to relocate joining our company? Talk about company environment advantages. |
| What do you know about the country you applied for? Talk about country advantages. |
| What is your preference of technologies (the ideal technology) you want to work with? Talk about current and preferred technical stack |
| What are your strengths and weaknesses?  Strength: never stop until I got my task finished. Weakness: sometimes I forget something I should do and I’m working on this issue by using a TODO list and do workouts. |
| What is your technology wish list? Python, Mongo DB, DevOps |
| Tell me about any achievement? Talk about NBE with 6 junior developers. |

|  |
| --- |
| HR questions to ask. |
| Will you provide a Visa sponsorship? |
| If you are satisfied about this interview, what’s the next step? |
| Do you have any comments for me? |

|  |
| --- |
| Technical questions to ask. |
| When start working in a task, is there any dependencies? |
| Based on which criteria my performance be measured? |
| Do you have a different technical stack based on projects type? |
| What is the software development methodology do you use now? |
| What is the source control do you use now? |

|  |
| --- |
| General |
| Factors to be considered when setting Application Architecture (technologies): - Third parties and integration with external systems  - Number of users  - Performance, Security, Client requirements, Server OS and specs - Modularity and Microservices, Handle generic request, Logs, Filters, ORM layer (business layer) |
| Hosts file: is a local plain text file that maps servers or hostnames to IP addresses, Its located in C:\Windows\System32\drivers\etc\hosts |
| TLS proxy: a layer between client and server to encrypt the request to secure connections to allow for additional networking services while protecting against denial-of-service attacks. |
| Denial-of-Service (DoS) attack: is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic, or sending it information that triggers a crash. |
| Metricbeat is a lightweight shipper that you can install on your servers to periodically collect metrics from the operating system and from services running on the server. Metricbeat takes the metrics and statistics that it collects and ships them to the output that you specify, such as Elasticsearch or Logstash.  *And you can visualize these metrics using Kibana.* |
| Heartbeat mechanism: monitors the connection between a manager and an agent and automates the cleanup procedure when the connection is lost. This enables both the manager and the agent to release resources that were allocated for maintaining the connection. |
| SDLC: Requirement gathering and analysis, Design, Implementation or coding, Testing, Deployment, Maintenance:  - Project preparation discuss resources and the skills needed, apply tranings or self-study if needed, define the sprint period and the story points count, define the branching strategy (pdf)  - Kick-Off meeting (Project overview, technical stack, source control, project management tool, hosting platform, deployment approach, etc…)  - All team members MUST understand the project business.  - Having a standup everyday for 30 minutes  - Having a grooming meeting (to be recorded) between all the project teams including the business team to set priorities of backlog tickes and clarify and estimate tickets, the estimation include the development and testing efforts  - One of ticket\task readiness factors is to have a test cases so that the developer can convert it to unit tests, that should be runned every time we do a deployment.  - Any Commit should have a clear comment that include the ticket number also.  - Integrate Github with Jira to show the created branch, merge or pull request for each ticket  - Integrate Jira with WiKI (which contains all project docs and also the grooming recordings)  - Retrospective meetings: (every 2 weeeks or in general after every sprint) are a great opportunity for people to come together to talk about what went well, what didn’t go so well, and what will be changed for the next sprint, and agree on some action items (decisions) to solve that <https://ideaboardz.com/for/Sprint%2035%20Retro/3906034>  Workflow:  - Create a new Jira Board that define the tickets workflow: TODO > In Progress > Blocked > Code Review > QA > QA Sign Off > UAT > Done  - DB objects should be added to source control, by creating a DataBase project in Visual Studio  - Release branch should have some constrains so only the release manager person is the only person who can approve the changes  - Once you have a ticket, create a feature branch (feature/AS-1234) and start develop  - Once you finish > create Pull Request from your feature branch to develop branch  - Once merged to develop, the jenkins job for build and deploy will be auto-triggered to deploy to the QA environment  - Once we have the QA SignOff, merge between develop and release  - Once the merge done, it will trigger the jenkins job responsible to deploy to UAT  - Once deployed to UAT, the business team should validate the ticket  - Once we have the business team SignOff for all sprint tickets that will be included in the release, we run the jenkins job to deploy to prod from release branch  - Once the prod deployment done, the QA should run the regression test  - Once we have the prod SignOff, then merge from release to master and develop branches |
| C# memory management: C# employs automatic memory management which implemented by a *garbage collector*, The garbage collector checks to see if there are any objects in the heap that are no longer being used by the application. If such objects exist, then the memory used by these objects can be reclaimed. (If no more memory is available for the heap, then the new operator throws an OutOfMemoryException).  *The managed heap*: When an application calls the new operator to create an object, there may not be enough address space left in the region to allocate to the object. The heap detects this by adding the size of the new object to NextObjectPinter. If NextObjectPinter is beyond the end of the address space region, then the heap is full and a collection must be performed. |
| Web Farm: is a group of two or more web servers (or nodes) that host multiple instances of an app. When requests from users arrive to a web farm, a load balancer distributes the requests to the web farm's nodes, so web farm improve Reliability/Availability, Capacity/Performance, Scalability, **Maintainability** |
| CRM: stands for Customer Relationship Management, a CRM system allows businesses to manage business relationships and the data and information associated with them. |
| CLI programming: CLI stands for Command Line Interface, Interpreter or Input which is a command line program that accepts text input to execute operating system functions. |
| Micro services: is an architectural style that structures an application as a collection of services that are:  Highly maintainable and testable, Loosely coupled, Independently deployable, Organized around business capabilities and  Owned by a small team. |
| Modularity: is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality. |
| Difference between client-side and server-side: the server side scripting involves server for its processing. ... The client-side script executes the code to the client side which is visible to the users while a server-side script is executed in the server end which users cannot see. |
| Service Account is a user account that is created explicitly to provide a security context for services running on Windows Server operating systems. The security context determines the service's ability to access local and network resources. The Windows operating systems rely on services to run various features. |
| Visual Studio CI/CD > Continuous Integration and Continuous Delivery: is a continuous delivery tool provided by TFS in visual studio which makes it simple to automate and stay up to date on your DevOps pipeline for ASP.NET and other projects targeting Azure. |
| Jenkins: is a free and open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. |
| Rolling deployment  allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources.  *Better to keep the app as stateless for rolling deployment* Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic. |
| Difference between Ansible and Terraform and PowerShell:  Terraform used to provision and configure infrastructure components, its written with (HCL) HashiCorp Configuration Language  Ansible used for deploy, install and manage the your application, its written with (YAML)  PowerShell is a Microsoft framework for automating tasks using a command-line shell and an associated scripting language. |
| Docker: (developer share a copy of deployment server to tester to avoid issues because environment difference)  is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. |
| $$ Kubernates: |
| Relational and non-relational databases: A relational database uses SQL (Structured Query Language) which storing data using tables, columns and rows but the non-relational databases (NoSQL) are document-oriented. Instead of using tables, these documents allow us to store unstructured JSON data in a single document. |
| How does the internet work: The Internet is made up of a massive network of specialized computers called routers. Each router's job is to know how to move packets along from their source to their destination. A packet will have moved through multiple routers during its journey. When a packet moves from one router to the next, it's called a hop. |
| Task implementation steps:  - Defining the problem clearly  - Breaking the problem down into small, simple parts  - Define the solution for each part of the problem  - Implementing the solution  - Making it efficient (eventually) |
| OLAP (Online Analytical Processing) queries: OLAP is a category of software that allows users to analyze information from multiple database systems at the same time. It is a technology that enables analysts to extract and view business data from different points of view. OLAP stands for Online Analytical Processing.  Analysts frequently need to group, aggregate and join data. These operations in relational databases are resource intensive. With OLAP data can be pre-calculated and pre-aggregated, making analysis faster.  OLAP databases are divided into one or more cubes. The cubes are designed in such a way that creating and viewing reports become easy. |
| Basic distributed computing concepts: A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another |
| Web Socket: a protocol enables interaction between a web browser (or other client application) and a web server with lower overhead, exchange messages between client and server in server use TcpListner and in client use WebSocket |
| Serverless Computing: is a cloud-computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity. |
| Scalable systems: is a system that is able to maintain its effectiveness throughout even if there are additions or expansions in aspects such as resources or the number of end users without disrupting its functionality then it can be called a scalable system. |
| A/B testing: also known as split testing or bucket testing, is a method of comparing two versions of a webpage or app against each other to determine which one performs better. |
| Agile environment: a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams and Client/End user. |
| Agile Scrum: is one of the implementations of agile methodology. In which incremental builds are delivered to the customer in every two to three weeks' time. it has a 3 key roles: Product Owner, Scrum Master, Development Team  *Sprint > EPIC > User Story > Feature > Task*  \* Every day a scrum meeting (stand-up meeting) for 15 minutes to discuss: What’s done yesterday? What will be done today? What’s your obstacles?  Scrum Master is the person who ensures that the team adheres to scrum theory, practices, and rules and protects the team by doing anything possible to help the team perform at the highest level. This may include removing impediments, facilitating meetings, and helping the Product Owner groom the backlog. |
| Code Quality Factors: Code Reviews, Readability, Maintainability, Code Documentation, Best Practice coding, follow a consistent style, Unit Testing so it should be testable by building a good software structure with less coupling. |
| Coding Standards: there are some tools like SonarLint and TSLint. |
| $$ SLA Service Level Agreement: |
| Quality Gates: Definitions for A Quality Gate is a special milestone in a software project Quality Gates are located before a phase that is strongly dependent on the outcome of a previous phase They are especially useful between phases. Formal sign-off and acceptance are mandatory at each gate.  >> Putting some measures to be checked after every phase of application |
| CI/CD:  CI (establish a consistent and automated way to build, package, and test applications, each change in code triggers an automated build-and-test sequence for the given project)  CD (Package the build output files to deploy to server) |
| CI/CD Pipelines: helps you automate steps in your software delivery process, such as initiating code builds, running automated tests, and deploying to a staging or production environment.  Also you can inject another layers inside these steps like a layer for checking coding style |
| Difference between bat and powershell scripts: Batch files are generally executed by CMD.exe; PowerShell is executed by PowerShell.exe. PowerShell includes extra features and commands that aren't in cmd.exe |
| Regression Test: A test that was written when a bug was fixed. It ensures that this specific bug will not occur again. |
| Sanity Test: is done to check the new functionality/bugs have been fixed |
| Integration Test: Test the correct inter-operation of multiple subsystems. There is whole spectrum there, from testing integration between two classes, to testing integration with the production environment. |
| Smoke Test: also known as “Build Verification Testing”, is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The result of this testing is used to decide if a build is stable enough to proceed with further testing. |
| Testing Pyramid (higher to lower cost)  UI  Service(API)  Unit Test (smaller components) |
| $$ Code Smell |
| $ Advantages of ORM: reduce query repetition, take the benefit of OOP to build a maintainable code |
| $ DisAdvantages of ORM: performance |
| Syntactic Sugar: means that the new features of the language are not really new. Instead, they are a nicer syntax for something existing/ You could do exactly the same by writing something different in the old version. ... Anything which is simplified is a syntactic sugar. |
| Cross-site request forgery (CSRF): also known as one-click attack or session riding and abbreviated as CSRF, is a type of malicious [exploit](https://en.wikipedia.org/wiki/Exploit_(computer_security)) of a [website](https://en.wikipedia.org/wiki/Website) where unauthorized commands are transmitted from a [user](https://en.wikipedia.org/wiki/User_(computing)) that the web application trusts.  There are many ways in which a malicious website can transmit such commands; specially-crafted image tags, hidden forms, and [JavaScript](https://en.wikipedia.org/wiki/JavaScript) XMLHttpRequests, for example, can all work without the user's interaction or even knowledge. Unlike [cross-site scripting](https://en.wikipedia.org/wiki/Cross-site_scripting) (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.  To prevent CSRF: inject an authentication token in every request to be validated in middleware |
| Cross-Site Scripting (XSS) Attacks: are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. |
| Hadoop: is a software technology designed for storing and processing large volumes of data distributed across a cluster of commodity servers and commodity storage. |
| Rabbit MQ: gave slightly better performance than MSMQ, but both were comprehensively out performed by ZeroMQ. If performance is your main criteria, you should definitely look at ZeroMQ.  MSMQ: is a simple store-and-forward queue. It doesn't provide any messaging patterns, such as pub/sub, or routing. For anything beyond simple point-to-point messaging you'd probably want to use a service bus library such as NServiceBus or MassTransit on top of MSMQ.  Rabbit MQ is a sophisticated server product that provides complex messaging patterns, topics and routing out-of-the-box. You also get centralized management and DR, something you'd have to implement yourself if you chose MSMQ. Rabbit MQ Workflow: > The producer publishes a message to the exchange.  > The exchange receives the message and is now responsible for the routing of the message.  > Binding must be set up between the queue and the exchange. In this case, we have bindings to two different queues from the exchange.  > The exchange routes the message into the queues. The messages stay in the queue until they are handled by a consumer.  > The consumer handles the message. |
| Rabbit MQ Cluster: is a logical grouping of one or several nodes, each sharing users, virtual hosts, queues, exchanges, bindings, runtime parameters and other distributed state. |
| $$ Separation of Concern SOC: |
| $$ AWS > ACS: |
| $$ AWS > S3: |
| Change-Advisory Board (CAB) delivers support to a [change-management](https://en.wikipedia.org/wiki/Change_management) team by advising on requested changes, assisting in the assessment and prioritization of changes. |
| Grooming (or refinement) is a meeting of the Scrum team in which the product backlog items are discussed and the next sprint planning is prepared. |
| Data warehouses are used for analytical purposes and business reporting. Data warehouses typically store historical data by integrating copies of transaction data from disparate sources Data warehouse is a system that stores data from a company's operational databases as well as external sources. Data warehouse platforms are different from operational databases because they store historical information, making it easier for business leaders to analyze data over a specific period of time.  *Also to enhance querying performance by reducing the number of records after moving the historical records to the warehousing DB* |
| Mortgage Release is where you, the homeowner, voluntarily transfer the ownership of your property to the owner of your mortgage in exchange for a release from your mortgage loan and payments |
| Data models define how data is connected to each other and how they are processed and stored inside the system. |
| INFO: Requests that responded live are Syncronous requests and the other are Asyncronous requests. |
| Stateless Application is an application program that does not save client data generated in one session for use in the next session with that client.  Stateful Application saves data about each client session and uses that data the next time the client makes a request. |
| SSO: |

|  |  |
| --- | --- |
| HTML | |
| DOM: (Document Object Model) represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content. | **Web storage types (session storage and local storage) > difference:**  window.localStorage**:** stores data with no expiration date, data stored per origin (a combination of protocol, host name, and port number)  window.sessionStorage**:** stores data for one session (data is lost when the browser tab is closed)  \* so, If you want to store a different data for the same app while open 2 tabs you should use session storage. |
| Difference between UL & OL: UL means "unordered list". OL means "ordered list". UL gets you bullet points. OL gets you numbers | **How to group a form elements:** Field set & Legend |
| What is input types? Button, date, email, file, submit, radio, checkbox, etc… | **HTML5 tags:** like article, footer, section, etc |
| Difference between .attr() & .prop(): prop has no value (checked, disabled, etc...)  Attributes are physical written in HTML but properties not |  |
| $$ XHTML: | |
| $$ XSL: | |
| $$ XSLT: used as a styled template to view xml data in brawser | |
| $$ XPath & XQuery: | |
| INFO: we can transform (XML and XSL\XSLT) to XHTML in client side using JS or in server side | |
|  | |

|  |  |
| --- | --- |
| CSS | |
| Bootstrap: is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation and other interface components. | **What’s the difference between display: block, inline and inline-block:** display: inline-block allows to set a width and height on the element. Also, with display: inline-block , the top and bottom margins/paddings are respected, but with display: inline they are not. |
| CSS selectors: <https://www.w3schools.com/cssref/css_selectors.asp> | **What is the different ways to apply CSS on element:** Inline, Internal (in style tag) or External |
|  | **How select the last child of an element:**  element > \*:last-child {} |
| How select nth child of an element: element:nth-child(n) | **Coloring:** (color name, HEX color and RGB) |
| What is new in CSS3:  Selectors: can use regular expressions formula (^ starts with, $ ends with, \* contains, etc...)  Rounded Corners and Border Image: using border-radius and border-image  Box Shadow and Text Shadow: using box-shadow and text-shadow  RGPA colors with opacity. | |

|  |  |
| --- | --- |
| JS & JQuery | |
| Syntax of document ready $(document).ready(function(){}); and $ what is the different ways? | **How to get\set element text\html:** .text() .html() |
| Document Ready Event in JS:  document.addEventListener("DOMContentLoaded", function () {}); | **Difference between Array.map() & Array.forEach:**  arr.forEach((item, index) => {return item++});  forEach() changes the original array, whereas map() returns a new array, without mutating the original array. |
| INFO: Javascript has only one thread but the browser has many | **INFO:** you can stop var xx = setInterval(fun(), interval) by clearInterval(xx); |
| What’s the different ways to bind event on HTML element:  JS: element.addEventListener(event, function, useCapture);  JQ: $(element).eventName(function(){}); or $(element).on(“eventName”, function(){}); so the difference is on() is binding event to element dynamically using below: $(parentElement).on(‘eventName’, ‘dynamicElemnt’, function(){}); | **Difference between bind() \* live**():  bind() attachs events to elements that exist or match the selector at the time the call is made. Any elements created afterwards or that match going forward because the class was changed, will not fire the bound event.  .live() works for existing and future matching elements. |
| JQ effects methods: (animate(), fadein(), fadeout(), etc..) | **Difference between parent(), children() and siblings():  Siblings:** are all elements in the same level and same parent |
| DOM Model: is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document. | **Difference between null and undefined**: undefined means a variable has been declared but has not yet been assigned a value. On the other hand, null is an assignment value. |
| Browser Indexed DB: is a new HTML5 concept to store the data inside user's browser. indexeddb is more power than local storage and useful for applications that requires to store large amount of the data. > It stores key-pair values > It is not a relational database > IndexedDB API is mostly asynchronous > it is not a structured query language > it has supported to access the data from same domain | |
| Abort Controller: interface represents a controller object that allows you to abort one or more Web requests as and when desired.  const abortController = new AbortController();  abortController.abort(); | |
| Prototypal Inheritance in JS: Each object has a private property which holds a link to another object called its prototype.  function Person(first, last, age, eyecolor) {   this.firstName = first;   this.lastName = last;  } Person.prototype.nationality = "English"; | |
| How to remove binding from element:  JS: element. detachEvent();  JQ: $(element).unbind(“event name”); | **What is preventDefault():** itcancels the event’s default behavior if it is cancelable, meaning that the default action that belongs to the event will not occur. |
| How to remove an element from DOM: $(element). remove(); | **How to add\remove class from element:**  $(element). addClass().removeClass(); |
| Keyboard events:  keyup() – Fire when user releases a key on the keyboard.  keydown() – Fire when user presses a key on the keyboard.  keypress() – Fire when user presses a key on the keyboard.  Keypress respects case sensitivity when accessing char ASCII code | **String operations:**  **JS:** length, indexOf(‘xx’), indexOf(‘xx’, startIndex), lastIndexOf(‘xx’), search(‘xx’) |
|  | **What is array.slice():** returns the selected elements in an array, as a new array object. |
| How to apply styles on element using JQuery: .css() | **Best performance of JS Loops**: a standard for loop with length caching:  for (var i = 0, len = myArray.length; i < len; i++) {  } |
| Comments: // /\* \*/ | **Inheritance in JS:** using prototype |
| $ Regular Expressions: | **JavaScript ecosystem:** is a collection of software packages, libraries, and other resources that facilitate development as they integrate with each other. Those tools are created by different developers and providers – for example, the React library is powered by Facebook, while the Angular framework was created by Google, and Vue.js was designed by an independent developer. |
| Difference between var, let and const:  var can be updated and re-declared within its scope;  let can be updated but not re-declared;  const can neither be updated nor re-declared.  They are all hoisted to the top of their scope but while var variables are initialized with undefined, let and const variables are not initialized. | **Difference between Bind, Call and Apply:** all of it to invoke a function, use .call() or .apply() when you want to invoke the function immediately, and modify the context. Call/apply call the function immediately, whereas bind returns a function that, when later executed, will have the correct context set for calling the original function.  var person1 = {name: 'Marvin', age: 42, size: '2xM'};  **var** sayHello = **function**(){  alert('Hello, ' + **this**.name);  };  sayHello.call(person1);  sayHello.apply(person1);  \* the 1st parameter should be the scope object (this)  \* **What if you need to pass parameters to the function while invoking it** you should use .call() but if you have many parameters (arguments) then you should use .apply() because it expects the 2nd parameter as array of arguments. |
| Implicit and Explicit binding: Implicit when this keyword presented inside code but Explicit when this keyword explicitly passed by .call(), .apply() or .bind() | **This and Context:** this is just the current context (caller) |
| Strict mode: introduced in ES5, it throws reference error when using variable before declaring it. | **Variable scope in JS:** JavaScript has two scopes – global and local. Any variable declared outside of a function belongs to the global scope, and is therefore accessible from anywhere in your code. Each function has its own scope, and any variable declared within that function is only accessible from that function and any nested functions. |
| Event delegation in JS: is adding event listener to parent element instead of adding to every child and inside this event using event.target you can detect which child who fired the event.  And using event.currentTarget to access the parent element | **Event propagation:** is a mechanism that defines how events propagate or travel through the DOM tree to arrive at its target and what happens to it afterward.  \* when you have a nested element(s) which assigned the same event (ex, click) so when you click on any one the other events fire so event.stopPropagation() will stop propagating event to other elements.  **> It has 2 types:** Capturing: Spreading event inside.  Bubbling: Spreading event outside. |
| $ How many 8-bit in text to be stored in Utf-8? | **When shouldn’t use arrow function:** when you want to access something from this (caller) |
| Difference between Reference Error and Undefined: in ES6 when using undeclared variable it will throw reference error but in earlier versions of JS will output undefined | **INFO:** any undeclared variable (without var keyword) becomes global |
| $ JS Versions and difference:  ES Standard, ES2015, ES6 and ES Next | **Let keyword:** make the variable block scope not function scope, if you defined var inside loop it will be accessible to its function but let wont be.  var x = function(){  for(var x = 0; x < 2; x++){  let y = 4;  }  console.log(y);  } if you run x() >> output : ReferenceError: y is not defined but if you use var instead of let >> output : 4 |
| Immutable Variables: are variables whose value cannot be modified once assigned (const) | **Difference between function declaration and function expression:**  Function declaration like function foo(){ … }  Function expression like var foo = function(){ … }  **\*** Function declaration is hoisted but expression isn’t. |
| Ternary operator (? & :) like this condition ? true : false | **Null Conditional Operator:** object?.property ?? defaultvalue |
| INFO: you can dynamically validate function parameters using arguments keyword | **DOM reflow**: a web browser process for re-calculating the positions and geometries of elements in the document, for the purpose of re-rendering part or all of the document. |
| INFO: let and const are not hoisted because it provide a block scope | **INFO:** string can be converted to number by putting + before it: var x = ‘23’; var y = +x; // 23 |
| INFO: you can convert number to string by putting ‘’ + befor it: var x = 23; var y = ‘’ + x; // “23” |  |
| Caching Client Side: Caching is used for web performance optimization since the configuration and server connectivity data are usually stored in *Cookies* but the popular data in web storage (*sessionStorage and localStorage*)  Cookie: (Cookies are data, stored in small text files, on your computer) stores data that has to be sent back to the server with subsequent requests. There are some libraries used in caching: Modenizr and Redis | |
| INFO: let x = 0, isTrue = false; if(x) isTrue = true; >> still false | |
| Syntactic Sugar: is designed to make things easier to read or to express. It makes the language "sweeter" for human use: things can be expressed more clearly, more concisely, or in an alternative style that some may prefer. | |
| What is Closure in JS: means writing a nested functions in which Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function) | |
| INFO: instead of using callback to handle response and error, you can use promises by .then(handleResponseFnc).catch(handleErrorFnc) | |
| Promises in JS: The Promise object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value, *it has 3 states*:   * *pending*: initial state, neither fulfilled nor rejected. * *fulfilled*: meaning that the operation completed successfully.   *rejected*: meaning that the operation failed. const myPromise = (new Promise(myExecutorFunc)).then(handleFulfilledA,handleRejectedA)  .then(handleFulfilledB,handleRejectedB)  .then(handleFulfilledC,handleRejectedC); | |
| $ Difference between promises.All & promises.Each:  The Promise.all() method returns a single [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) that fulfills when all of the promises passed as an iterable have been fulfilled or when the iterable contains no promises or when the iterable contains promises that have been fulfilled and non-promises that have been returned. It rejects with the reason of the first promise that rejects, or with the error caught by the first argument if that argument has caught an error inside it using try/catch/throw blocks.  const promise1 = Promise.resolve(3);  const promise2 = 42;  const promise3 = new Promise((resolve, reject) => {  setTimeout(resolve, 100, 'foo');  });  Promise.all([promise1, promise2, promise3]).then((values) => {  console.log(values);  }); | |
| What is Async and Await : async/await simply *gives you a synchronous feel to asynchronous code*. It's a very elegant form of syntactical sugar, it require a transpiler to be read be browser (babel). await: block code execution until finish it, and it should be used if the next step depends on the previous operation result.  while (!value) {  const intermediate = await operation1();  value = await operation2(intermediate);  } | |
| INFO: Marking function as async makes it also return a promise. This is done automatically for you, so you don't need to return a promise explicitly. | |
| INFO: the await keyword is only valid inside async functions | |
| What is Array.reduce : used to reduce the array to a single value and executes a provided function for each value of the array (from left-to-right) and the return value of the function is stored in an accumulator.  array.reduce( function(accumulator, currentValue, currentIndex, arr), initialValue ) | |
| $ What is the output of the below code: How setTimeout function works ? for( I = 0; I < 5; i++){  setTimeout( ()=>{console.log(i)});  } | |
| INFO: use string literal (`) to avoid conflicts of single\double quotes of string and to bind a value of variable inside string  var x = 3;  var y = `this is x = ${x}’; | |
| Bugsnag: [Error Monitoring & Reporting Tool for App Stability](https://www.bugsnag.com/) | |
| Sentry: Used to capture Javascript errors | |
| HotJar: Monitor and fix the UX errors of your application. | |
| Difference between event target and currentTarget : target is the element that triggered the event (e.g., the user clicked on) currentTarget is the element that the event listener is attached to. | |
| How to authenticate API from frontend: Using JWT to be sent in every request header, a JWT is a token that is issued by the server. The token has a JSON payload that contains information specific to the user. This token can be used by clients when talking to APIs (by sending it along as an HTTP header) so that the APIs can identify the user represented by the token, and take user specific action.  The JWT needs to be stored inside an httpOnly cookie, a special kind of cookie that’s only sent in HTTP requests to the server | |
| JWT (JSON Web Token): way for securely transmitting information between parties as a base64 encoded JSON object, consist of three parts separated by dots (header.payload.signature)  So Header contains the algorithm name and type (JWT)  Payload contains some useful data (user claims) like iss (issuer), exp (expiration time), sub (subject), aud (audience), and others.  Signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is. > A JWT is not encrypted. It is based64 encoded and signed. So anyone can decode the token and use its data. | |
| $ What is Selenium Testing: | |
| $ What is End To End Testing: The entire application is tested for critical functionalities such as communicating with the other systems, interfaces, database, network, and other applications. | |
| Functional Programming: define the problem as a small functions that its output depend on the parameters .. it acheives modularity and may be related to Business Rule Engine (BRE) => may be :D | |
| Reactive Programming (RP): (is programming with asynchronous data streams), (It is an Asynchronous Programming concept around data streams.), (is writing code that define how to react to changes) and (is a programming paradigm oriented around data flows and the propagation of change.)  var x = 10;  var y = 20;  let z = x + y;  console.log(z);  x = 100;  y = 200;  console.log(z);  So here the variable Z will not be changed and RP is to write code that define how to react to changes by definining data\code as streams: import {Observable} from 'rxjs-es'; Using Observable represented in RXJS and to subscribe to specific variables.  > Both promise and observables are built to solve problems around async (to avoid “callback hell”).  > An observer is object literal with next(), error() and complete() functions. In the above example, the observer is the object literal we pass into our .subscribe() method. | |
| Callback Hell: is having a lot of nested callback functions, Each inner callback is dependent on its parent (code becomes obscure or difficult to follow.)  So promises and observable (RXJS) fixed this problem. | |
| RXJS (Reactive Extensions for JavaScript): is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code. | |
| ErrorFirst Callback: is simply structuring the callback function by writing the first argument\parameter as error object and the subsequent arguments represent associated data. | |
| What is Hoisting: is moving variables and functions declarations (*not initializations*) to the top of their scope before code execution, hoisting has order while topping up > variable assignment, function declaration then variable declaration | |
| Difference between variable hoisting and function hoisting: Unlike variables, a function declaration doesn't just hoist the function's name. It also hoists the actual function definition. As you can see, the JavaScript interpreter allows you to use the function before the point at which it was declared in the source code. | |

|  |  |
| --- | --- |
| Typescript | |
| A clear difference between ECMA Script 5 & 6? ES6 added some features to ES5 like const, let , etc… | **Compilers\Transpilers to convert ECMA 6 to 5?** yes |
| INFO: Type (tsc -w) in terminal to switch on the auto-compile | **INFO**: Type (tsc --init) to add tsconfig.ts to project and configure (compileOnSave ) |
|  | **INFO**: Use Tuple to create an array with a static different types. |
| INFO: use enums in static variables grouping | **What is strict mode:** a new feature in ECMAScript 5 that allows you to place a program, or a function, in a “strict” operating context. This strict context prevents certain actions from being taken and throws more exceptions. |
| $ Why you prefer writing Typescript rather than Javascript? | |
| INFO: Always use let (controls the local scope of variable) | |
| INFO: Variable local\global scope (When use declaration, it will be local) | **INFO**: Block scope is simply writing a code inside {} |
| INFO: You can set readonly variable only in initialization or constructor, so the difference between Const and readonly is that you can’t initialize const without setting a value but readonly is applicable and set its value in constructor | **Arrow Function:** Lambda expression in Typescript: let Do = function (a, b){ return a\*b} 🡺 let Do = (a, b) => (a\*b)  Most important advantage of arrow function doesn’t bind to this |
| Difference between == and ===: == compare value but === compare both value and type | **Spread and rest operator**: to spread array items (ex. array spread by ...arr)  function showName(firstName, lastName, ...titles) {  alert( firstName + ' ' + lastName ); // Julius Caesar  alert( titles[0] ); // Consul  alert( titles[1] ); // Imperator  alert( titles.length ); // 2  }  showName("Julius", "Caesar", "Consul", "Imperator");  *The ...rest parameter (titles) must always be last one.* |
| INFO: Immediately invoked function expression (IFIE Module) to define function and call it in same time, also used to control conflicts if some var name duplicated in 2 JS files: ex: (function(){ //do something}()); | |
| INFO: => is abbreviation for (go through) | **Typescript mixed data type**: (ex. let x: number|string) |
| Typescript type alias: (ex. type int = number; let y: int) | **JS inheritance**: > Lec 2 > 1:15 min |
| Revealing Module Pattern: (every JS file is standalone (using export & import)) | **Use (System JS)**: as module loader to import a JS file inside HTML file. |
| INFO: To run a .ts file using node, use command (node filename.ts) | **INFO**: You can create a Typescript project from Visual Studio (HTML Application with Typescript) |
| Decorator: adding another fields\properties to a class without editing its code or inheritance using thatClass.prototype.newProperty = something | **INFO**: Loop through array by for (let n in array) to access index OR for (let n of array) to access object itself |
|  | |

|  |  |
| --- | --- |
| React.JS | |
| Difference between function component and class component: Function component preferred to be used when this component has no user interactions | **What is React mountNode**: is a DOM node which you are telling ReactDOM to render the component into. |
| $ React rendering in server side : (so the search engines can fetch and index all application pages) | |
| $ Styled Components: | **JSX:** (JavaScript Syntax XML) is writing JS code inside HTML. |
| $ Interview questions and answers: |  |
| INFO: You can simulate a response data by using website (https://jsonplaceholder.typicode.com) | **INFO**: When you pass a props to a component, you should define this props like below:  class X extends Component <any, any> // at least use any data type |
| INFO: you can’t use setState in constructor, just use this.state.x = value; | **What is Props:** Props is the shorthand for Properties in React. They are read-only components which must be kept pure i.e. immutable. They are always passed down from the parent to the child components throughout the application. A child component can never send a prop back to the parent component. This help in maintaining the unidirectional data flow and are generally used to render the dynamically generated data. |
| INFO: To install React Router > npm install --save react-router | **Difference between NPM and NPX:** NPX makes it easy to install and manage dependencies hosted in npm registry. It simplifies the process and provides a better for executables. |
| TS Webpack: is a tool that will bundle your code and optionally all of its dependencies into a single .js file  TS Babel: used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments  TS Gulp: is a cross-platform, streaming task runner that lets developers automate many development tasks. | **Webpack installation**:  - move index.html from build to src  - install all required modules??  - use HashRouter  - in tsconfig > remove "noEmit" and set "jsx" to "React" |
| INFO: componentDidMount will fire after componentWillReceiveProps as well | **React Virtual DOM:** In React, for every [DOM object](http://eloquentjavascript.net/13_dom.html), there is a corresponding “virtual DOM object.” A virtual DOM object is a *representation* of a DOM object (*like a lightweight copy*) which have also the same properties. So when you update some element in virtual DOM object it totally rerendered so react calculate the difference between the Real DOM and Virtual DOM and update only the changed objects into the Real DOM not all objects. |
| Jest Unit Testing: Jest was created by Facebook and is a testing framework to test JavaScript and React code.  Enzyme is a *JavaScript Testing utility* for React that makes it easier to assert, manipulate, and traverse your React Components' output. Enzyme, created by Airbnb, adds some great additional utility methods for rendering a component (or multiple components), finding elements, and interacting with elements. Ex expect(element).toBe(value); | |
| React Life Cycle Methods:  Constructor: to set up the initial state and other initial values.  Render: the method that actual outputs HTML to the DOM.  componentDidMount: method is called after the component is rendered; this is where you run statements that requires that the component is already placed in the DOM.  shouldComponentUpdate: method you can return a Boolean value that specifies whether React should continue with the rendering or not.  componentDidUpdate: method is called after the component is updated in the DOM.  componentWillUnmount: method is called when the component is about to be removed from the DOM. | |
| $ How React state works? | |
| $ What is subscribe? | |
| $ When you should unsubscribe? | |
| What is the property (key) is used for in React: Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside a repeater. | |
| Refs: is the short hand for References in React. It is an attribute which helps to store a reference to a particular React element or component, which will be returned by the components render configuration function. It is used to return references to a particular element or component returned by render(). They come in handy when we need DOM measurements or to add methods to the components. <input ref={snap => this.Snaps.inputUserName = snap} />  there are some instances when using Refs is useful, such as:  Managing focus, text selection, or media playback.  Triggering imperative animations.  Integrating with third-party DOM libraries. | |
| What is React Context: data is passed top-down (parent to child) via props, Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.  const MyContext = React.createContext(); | |
| Difference between state and props: state is mutable but props are immutable, props are variables passed to it by its parent component. State on the other hand is still variables, but directly initialized and managed by the component. The state can be initialized by props. | |
| Why is switch keyword used in React Router v4?The <switch> tag when in use matches the typed URL with the defined routes in sequential order. When the first match is found, it renders the specified route. Thereby bypassing the remaining routes. | |
| Is setState() async? Why? yes, it uses a callback function to fire after completely update state this.setState({}, () => {});  This is because setState alters the state and causes rerendering. This can be an expensive operation and making it synchronous might leave the browser unresponsive. | |
| $ React Hooks: | |
| $ React HOC: | |
|  | |
|  | |

|  |  |
| --- | --- |
| Redux | |
| Definition: A centralized data store that all representational screen reads from :  1. Component > call an Action  2. Action > dispatch action so reducers can update state.  3. Reducer > make a new copy of state and return the new state (Immutability).  4. Store > after reducer update the state, the Store makes sure that all connected components are aware.  5. React-Redux > detect if there is state change and call react again.  6. React > if the UI depends on props updated by state, it will update the UI. | |
| INFO: Redux is the most common React State Management Library, there is another libraries like Flux. | **Babel Transpiler**: is used to convert from ES6 to ES5. |
| WebPack: is used to compile\minify JS into browser. | **ESLint**: is used to make a compile time checks on code to figure out the code mistakes |
| Function Components: Use Stateless components when you don’t need (State, Refs, Lifecycle Methods and child functions). | **Container\Presentational Components**: Container Com. almost is actions but Presentational Com. almost is HTML |
| Dispatcher: is the method used to dispatch actions and trigger state changes to the store. | **INFO**: Redux has no Dispatcher because it has only one store. |
| INFO: Redux is all about 3 stages (Actions, Reducers and Store). | **Immutability**: is instead of change object’s data return a new object, and on the other side (Mutable in JS like Objects, Arrays and Functions) |
| INFO: I can handle Immutability using ES6 by (Object.assign or Spread Operator).  var target = Object.assign(target, source); | **INFO**: To avoid Mutate state you can enforce Immutability over you project by including (redux-immutable-state-invariant) library to warn you when doing by mistake. |
| INFO: Redux and Flux are React workflow methods |  |
| Difference between mapStateToProps and mapDispatchToProps:  *mapStateToProps*: used to connect the store state to component props (subscribe) *mapDispatchToProps*: used to bind the action to props | |
| Redux Reducer: is a *pure sync function* that takes the previous state and an action, and returns the next state. | |
| Since you should use a pure functions in Reducers, How to call api call inside it (Async) : by using [Action creators](https://redux.js.org/advanced/async-actions) to make async calls and dispatch actions on completion of api call (using .then(){ dispatch(…)} ) | |
| Redux Observable: is an RxJS-based middleware for Redux that allows developers to work with async actions. It's an alternative to redux-thunk and redux-saga. | |
| $ How to structure store data\schema: | |

|  |  |
| --- | --- |
| Ajax | |
| Ajax: Asynchronous JavaScript And XML | **XHR:** XML HTTP Request |
| XHR onreadystatechange: fires 5 times with different values of readystate as below:  0 > Request not initialized  1 > Server connection established  2 > Request received  3 > Processing request  4 > Request finished and response is ready | **Difference between Sync and ASync:**  Synchronously, in which the script stops and waits for the server to send back a reply before continuing. Asynchronously, in which the script allows the page to continue to be processed and handles the reply if and when it arrives. |
| HTTP Request Headers:  General: contains data like request url, method, status code, etc…  Request Header: data like connection (keep-alive), cookies, host, etc…  Response Header: data like Server, Cache-Control, Access-Control-Allow-Origin, etc… | |
| What is $.Ajax params: method, url, data, cache(boolean), async(boolean), timeout(millisecond), success(data, status, xhr), error(xhr, status, errorMessage) and complete(xhr, status) which run in both success and error states.  *dataType: is the data type of response*  *contentType: is the data type of request* | **What is XHR params:**  var xhttp = new XMLHttpRequest();  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  console.log(xhttp.responseText);  }  };  xhttp.open("GET", "filename", true);  xhttp.send(); |
| Difference between Web API & Web Service (asmx): | **INFO**: Ajax Request = HTTP request |
| HTTP call life cycle:  1. First the browser needs to translate www.google.com to an IP address, it contacts a DNS server to resolve the name.  2. Then browser will open a TCP connection to the IP address of www.domain.com and send a HTTP GET request over.  3. The server software will get this HTTP request. It will somehow generate a HTTP response and send that back trough the TCP connection. (This is server software dependent).  4. When the browser gets the response, it typically renders it on screen. | **JSON & XML:** JSON (JavaScript Object Notation) is a lightweight data-interchange format and it completely language independent. ... Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. |
| INFO: XML is much more difficult to parse than JSON. | **INFO**: JSON is parsed into a ready-to-use JavaScript object. |
| JSONP: is really a simple trick to overcome the XMLHttpRequest same domain policy, by calling url in <script> tag instead of making a XMLHttpRequest, so if we imagine this  script = document.createElement('script');  script.type = 'text/javascript';  script.src = 'http://www.someWebApiServer.com/some-data';  will return this  <script>  {['some string 1', 'some data', 'whatever data']}  </script>  So to be able to handle this response we path a callback parameter so the server will return data as below  script.src = 'http://www.someWebApiServer.com/some-data?callback=my\_callback';  will return  <script>  my\_callback({['some string 1', 'some data', 'whatever data']}); </script>  so this is the JSONP is simply pass a callback parameter when calling api from script tag to overcome domain policy. | |
|  | |
| INFO: Cookies are commonly used to identify the same user across multiple HTTP requests, called sessions. Therefore these cookies are called session cookies. | **What is Proxy**:Implementing IHTTPHandler. |
| HTTP Result Codes: <https://restfulapi.net/http-status-codes/>  1xx: Informational 2xx: Success  3xx: Redirection 4xx: Client Responsible Error (400 Bad Request, 401 Unauthorized, 404 Not Found and 405 Method Not Allowed) 5xx: Server Responsible Error (500 Internal Server Error) | |

|  |  |
| --- | --- |
| Web service & Web API | |
| Web service: typically offers a WSDL from which you can create client stubs automatically. | **Web Services**: are based on the SOAP protocol. |
| Web API: is a newer Microsoft framework which helps you to build REST based interfaces.  The response can be either JSON or XML, but there is no way to generate clients automatically because Web API does not offer a service description like the WSDL from Web Services. | **INFO**: Restful (Representational State Transfer) & Soap (Simple Object Access Protocol) |
|  |  |
| REST APIs: uses multiple standards like HTTP, JSON, URL, and XML while. | **SOAP APIs**: is largely based on HTTP and XML. |
| REST APIs: are more convenient with JavaScript and can be implemented easily as well. | **SOAP**: is based on XML with meta data. |
| $ Status Codes | **$ Difference between MVC Controller and Web API Controller:** |

|  |  |
| --- | --- |
| C#.NET && OOP | |
| Concrete class: is a class that has an implementation for all of its methods | **INFO**: Assembly > Namespace > Class > Props & Methods |
| Class is a blue print that you create objects from, so that you define props in class and assign these props values while instantiating it (creating an object), and also you need to make this assignment in class constructor for the mandatory props so that you will defintly pass it when you create object. | |
| AssemblyInfo.cs is containing all data related to your assembly like Title, Desc, CopyRight, TradeMark and Version | Reserved Bytes for data types: bool: 1, int: 4, string: 4, float: 4, double: 8, decimal: 16 |
| INFO: By default the C# compiler treat number as double so to prevent this we can add a suffix (f) to the number like this float number = 12.45f; | **INFO**: If you have to inherit from BaseClass and interface you should put the BaseClass first like this public class ChildClass : BaseClass, IClass (not the oposite) |
| How compiler deal with C# code?  The C# compiler compiles the C# source code into the module, which is finally converted into the assembly. The assembly contains the Intermediate Language (IL) code along with the metadata information about the assembly.  The common language runtime (CLR) works with the assembly. It loads the assembly and converts it into the native code to execute the assembly.  Then this native code is executed by the Operating system and the output will shows according to your requirement. | |
| INFO: Keep your code simple, and don't do over-engineering. | |
| Trusted Connection in Connection String If you specify either Trusted\_Connection=True; or Integrated Security=SSPI; or Integrated Security=true; in your connection string THEN you have Windows Authentication happening. Any user id= setting in the connection string will be ignored | |
| Why using static keyword: means something which cannot be instantiated. You cannot create an object of a static class and cannot access static members using an object | |
| Data-Transfer-Object (DTO) is used to exchange data between different parts of an application Domain objects (DO) can have logic (depending on whether you are using domain-driven design or have anemic data model) and they are usually related to the database structure. | |
| C# Extension method: allows you to add new methods in the existing class or in the structure without modifying the source code of the original type  \* should be implemented in a static class and the method should have this as 1st parameter followed by class name and object (binding parameter) | **Lock keyword:** (used in multithreaded applications to ensure that the current thread executes a block of code to completion without interruption by other threads.) |
| .Net Sitecore: is a leading digital experience software used by organisations globally to create seamless, personalised digital experiences. Sitecore is an integrated platform powered by . net CMS, commerce and digital marketing tools. | |
| Mock object: (simulate the behavior of a real object) | **SonarLint**: is a code analyzer. |
| HTTP Handlers: When a client makes a request for a resource located on the server in an ASP.NET application, each request is handled by the HTTP Handlers.  It is an extension based processor and responsible for fulfilling requests from a browser depending on file extensions. When HttpHandler receives any request from browser, it checks the extension to see if it can handle that request and performs some predefined steps to respond that request.  So your custom HTTPHandler should implement 2 members of IHTTPHandler interface:  ProcessRequest Used to catch and fulfill your response of Http Requests.  IsReusable To check the reusability of same instance handler with a new request of same type.  <add verb="\*" path="\*.text" type=" ExampleHandler.Handlerclass, ExampleHandler "/> | |
| $$ Big Data Types in .Net | |
| .Net Core Middleware: is nothing but a component (class) which is executed on every request in ASP.NET Core application. In the classic ASP.NET, HttpHandlers and HttpModules were part of request pipeline. Middleware is like HttpHandlers and HttpModules where both needs to be configured and executed in each request.  We can configure middleware in the Configure method of the Startup class using IApplicationBuilder instance.  The request pipeline can have multiple middleware components (classes), Each middleware component in the request pipeline is responsible for invoking the next component in the pipeline. | |
| What is OOP and 4 concepts:  OOP is a way of writing computer programs using the idea of "objects" to represent data and methods.  Inheritance: a class can “inherit” the characteristics of another general class.  Abstraction: (*pure class for deneric purpose*) define some common behavior that can be inherited by multiple subclasses, without implementing the entire class.  Encapsulation: binds the member function and data member into a single class  Polymorphism: This is the ability of an object to perform in a wide variety of ways, it has 2 types:  - Dynamic polymorphism (runtime time) > overriding (allows a subclass to override a specific implementation of a method already issued by one of its super-classes)  - Static polymorphism (compile time) > overloading (defining several methods at the same time with the same method name but different in number, type or order of parameters) | |
| Interface: is a collection of abstract methods and doesn’t have constructor | |
| Simple definition for Interface: for example the company needs a Developer with a specific skills Not need (Mohamed) so in any time the company may replace Mohamed with any other Developer that has these skills so the Interface here is the Developer Skills as a Contract. | |
| INFO: Interface objects should not have access modifiers in signature, just return type, name and parameters | |
| INFO: the class that implement interface should have the implemented methods as public | |
| INFO: static class should not have a constructor because it cannot be instantiated. | |
| INFO: override keyword is used in either implementing a abstract method in the base class or overriding a method in base class that has virtual in its signature. | |
| INFO: No difference between Person person1 = new Person{ Name = ""}; AND Person person1 = new Person(){ Name = ""}; | |
| Overloading can be acheived by difference in return type as long as we have a difference in parameters. | |
| Early Binding: is the static binding in which recognize and check the [methods](https://www.geeksforgeeks.org/c-methods/), or [properties](https://www.geeksforgeeks.org/c-properties/) *during compile time*. In this binding, the compiler already knows about what kind of object it is and what are the methods or properties it holds. | |
| Late Binding: is the dynamic binding in which the compiler does not know about what kind of object it is and what are the methods or properties it holds, here the objects are dynamic objects. The type of the object is decided on the bases of the data it holds on the right-hand side *during run-time*. | |
| INFO: Function overloading is based on number\types\order of parameters | |
| Difference between abstract and interface: interface is not a class. ... The main difference between them is that a class can implement more than one interface but can only inherit from one abstract class. Since C# doesn't support multiple inheritance, interfaces are used to implement multiple inheritance.  Another thing that interface can’t contain any implemented methods but abstract can have implemented methods and not implemented methods (with abstract in signature) so when a class implement an abstract it should override all its abstract methods.  Abstract class cannot be instantiated because at least one method has not been implemented. | **Difference between Class & Struct:** Structs are value types while classes are reference types. structs can be instantiated without using a new operator. A struct cannot inherit from another struct or class, and it cannot be the base of a class |
| Repository Pattern: is a mediate (abstraction) layer between the domain (business) layer and data mapping layer (ORM) so that you hide how the data is eventually stored or retrieved to and from the DB, so the main purpose here is to make your achitecture a loosly decopled of ORM | |
| INFO: When running code, every function reserve a memory location in the stack so any varaible inside this function should be stored in this memory location, so this is how the function scope variables are no longer available as soon as the compiler finished the function execution and go outside it because at that time it should be deleted from memory. | |
| INFO: The memory of Stack got deallocated as soon as the function ended but in Heap its still as long as the application is running. | |
| INFO: When any application start, the memory manager auto reserve a specific size in Stack for it, but the Heap size can vary during the runtime of the application (Heap is a free pool of memory or dynamic memory), so the Stack memory slots auto-cleared as soon as the function finished calling, but the heap memory cleaning is managed by the garbage collector. | |
| INFO: Memory Leak can happen when we inproperly use Heap so there is a lot of unused or unreferenced blocks in the Heap, but for Stack you may have StackOverFlow error | |
| INFO: You allocate momory in Heap whenever you use (new) keyword. | |
| INFO: Memory has different parts: Stack, Heap and Global, Global is used to save the global objects (not function scope) | |
| INFO: Any recursive function should have a condition to end the recursion. | |
| INFO: In mathematical expressions, Multiplication\devision has a priority before addition\substraction. | |
| Difference between value type and reference type: A Value Type holds the data within its own memory allocation (in stack) and a Reference Type contains a pointer (in stack) to another memory location that holds the real data (in heap).  Stack memory is auto deleted as soon as compiler go out of scope of this data, but Heap memory is managed by GC Garbage Collector.  Reference Type variables are stored in the heap and *its reference pointer stored in the stack* while Value Type variables are stored in the stack.  User objUser = new User()  {  UserID = 1,  Name = "Jeorge"  };  ChangeName(objUser);  Console.WriteLine(objUser.Name);// Jon Doe  public static void ChangeName(User user)  {  user.Name = "Jon Doe";  }  *But if we changed ChangeName function to*  public static void ChangeName(User user)  {  user = null;  }  Console.WriteLine(objUser.Name);// Jeorge | |
| Despite the String is a reference type it passed by value, What is the secret? Because a copy of its reference that’s passed by value | |
| Difference between Stack & Heap: Objects allocated on the stack are available only inside of a stack frame (execution of a method), while objects allocated on the heap can be accessed from anywhere | **What is Data Structure in C#:** is a particular way of organizing data in a computer so that it can be used effectively. |
| What is Collections?  are classes that specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. | |
| What is Generics? allow you to write a class or method that can work with any data type.  static void Swap<T>(ref T lhs, ref T rhs) {  T temp;  temp = lhs;  lhs = rhs;  rhs = temp;  } | |
| Generics: reduce code redunduncy and increase performance, instead of using objects that make us do boxing and unboxing, we use generics to pass the type:  public calss GenericClass<T> where T : IComparable{  public T Add(T input1, T input2){    }  }  So, here we have applied generics constraint for T to accept any type that implement IComparable, also we can add more constraints (separated by comma) | |
| What is Generic Collections? A generic collection is strongly typed (you can store one type of objects into it) so that we can eliminate runtime type mismatches, it improves the performance by avoiding boxing and unboxing.  class Check<T> {  // Gerefic function to compare all data types  public bool Compare(T var1, T var2)  {  if (var1.Equals(var2))  {  return true;  }  else  {  return false;  }  }  }  Check<int> obj1 = new Check<int>();  bool intResult = obj1.Compare(2, 3); | |
| Event handlers: are nothing more than methods that are invoked through delegates | **Delegates:** are nothing but a function pointer |
| Delegates: is an agreement\contract between publisher and subscriber, so simply if you want to do some generic functionality right after finishing executing some task then implement a Delegate Evenet for example:  \* so the idea here is that you have the ability to add any other functionality to be executed after finishing video decoder,  public class MyApp{  public void MyMethod(){  VideoDecoder decoder = new VideoDecoder(); //publisher  Notifier notifier = new Notifier(); //subscriber  // let the notifier subscribe to the VideDecoded event  decoder.VideoDecoded += notifier.OnVideoDecoded;  decoder.Decode();  }  public class VideoDecoder{ // publisher  public event EventHandler VideoDecoded;  public void Decode(){  // decoding...  }  protected void OnVideoDecoded()  {  if (VideoDecoded != null)  VideoDecoded(this, EventArgs.Empty);  }  }  public class Notifier{ // subscriber  public void OnVideoDecoded()  {  SendMail();  }    public void SendMail(){  // sending mail...  }  }  } | |
| Who to call several methods in one shot: using Task await (asynchronous threading)  Or using delegate  delegate int deleg(int x);  deleg delegCall = function1Name;  delegCall += function2Name;  delegCall(params if there) | |
| Difference between ref and out keywords:  ref tells the compiler that the object is initialized before entering the function, while out tells the compiler that the object will be initialized inside the function.  \* ref is two-ways, out is out-only.  \* method must set the out variable before returning. | |
| Difference between == and string.Equals: string.Equals compares the content itself but == compares the reference address | **Difference between string and stringBuilder:** string Builder represent a mutable string of characters. ... So String objects are immutable but String Builder is the mutable string type. It will not create a new modified instance of the current string object but do the modifications in the existing string object. |
| Difference between var and dynamic: var is a statically typed variable. It results in a strongly typed variable, in other words the data type of these variables are inferred at compile time. ... dynamic are dynamically typed variables. This means, their type is inferred at run-time and not the compile time in contrast to var type. | **Difference between & and &&:** && evaluates the left side of the operation, if it's true, it continues and evaluates the right side. & is bitwise AND operator comparing bits of each operand. |
| Difference between sync and async: Synchronization means two or more operations happen sequentially. Asynchronous means two or more operations are running in different contexts (thread) so that they can run concurrently and do not block each other. | |
| Why use async await: Suppose that you have a method with 3 parts, and the 3rd part need the result of the 1st, also you don't want the 2nd to wait for the 1st:  public async Task MyMethodAsync() {  Task<int> longRunningTask = LongRunningOperationAsync();  // independent work which doesn't need the result of LongRunningOperationAsync can be done here  //and now we call await on the task  int result = await longRunningTask;  //use the result Console.WriteLine(result);  } | |
| What is Refactoring: is the process of changing the code structure after we complete the writing of the code to increase readability and easy maintenance of the code. Refactoring is done by changing the internal structure of the code without changing the external behavior of the code block. | |
| HangFire: An easy way to perform background processing in .NET and .NET Core applications. | |
| INFO: Abstract class doesn't allow us to instantiate it but inherit. | |
| Abstract class: what if you have 2 different class with the same target like SQLDataAccess and OracleDataAccess, then the optimum way to design it like below:  like this we have implemented the common methods only one time and give the ability to the child classes to override the methods that they have a difference in implementation.  Also, if you want to give the ability (as optional) to the child classes to override the LoadConnectionString method just put (virtual) keywaord in its signature:  public abstract class DataAccess{  public void LoadConnectionString(){  ....  }  public abstract void LoadData();  public abstract void SaveData();  }  public class SQLDataAccess: DataAccess{  public override void LoadData(){  }  public override void SaveData(){  }  }  public class OracleDataAccess: DataAccess{  public override void LoadData(){  }  public override void SaveData(){  }  } | |
| INFO: If you have more than 1 constructor and you want to call one from another (like that the base constructor will run first):  public class Customer{  public int Id;  public string Name;  public Order \_order;  public Customer(){  this.Id = 1;  }  public Customer(string name): this(){  this.Name = name;  this.\_order = new Order{  Id = this.Id,  Name = this.Name  };  }  } | |
| Authentication Types: Anonymous, Basic, Forms and Windows > difference?  ASP.NET provides two main ways to secure your web applications. They are - Windows authentication and Forms authentication. Windows authentication uses windows users names and passwords to authenticate them where as Forms authentication typically uses user ids and passwords stored in some database. | **Using operator:** The using statement defines a scope at the end of which an object will be disposed. The using directive creates an alias for a namespace or imports types defined in other namespaces. The using static directive imports the members of a single class. |
| ?? When to use Struct: |  |
| .NET Authentication: <https://dotnettutorials.net/lesson/token-based-authentication-web-api/>  using Microsoft.Owin.Security.OAuth;  In startup.Auth.cs  static Startup() {  OAuthOptions = new OAuthAuthorizationServerOptions() {  AllowInsecureHttp = true,  TokenEndpointPath = new PathString("/token"),  AccessTokenExpireTimeSpan = TimeSpan.FromDays(360),  Provider = new OAuthAppProvider()  };  } > By creating a class that inherits from OAuthAuthorizationServerProvider to override 2 methods: ValidateClientAuthentication:  GrantResourceOwnerCredentials: in which validate the client credentials (i.e. username and password). If it found the credentials are valid then create ClaimsIdentity object to set some user’s data\claims using ClaimType enums including ClaimType.Role to be used in Authenticate(Role = “”), then only it generates the access token. The client then using this access token can access the authorized resources from the Resource Server. > By calling /token and pass the data {username: ‘’, password: ‘’, grant\_type = 'password'} to authenticate user and return his data. | |
| INFO: If you want to segregate two interfaces without repeat members you can do this:  public interface IProduct{  void A();  void B();  }  public interface IClothes{  void C();  }  public class Dress: IClothes{  } | |
| Complexity analysis: is a rough approximation of the number of steps necessary to execute an algorithm. | **ADO.NET:** is a set of classes (a framework) to interact with data sources such as databases and XML files. It allows us to connect to underlying data or databases. It has classes and methods to retrieve and manipulate data. |
| $ What is cloud-computing systems: | **$ (IOC) Inventory of control (.NET Core):** |
| $ Difference between .Net and .Net Core: The .NET Framework supports Windows and Web applications, .NET Core is the new cross-platform and open-source .NET framework to develop applications for all major operating system including Mac, Linux, and Windows.. and it also provides the middleware layer.  And Web.Config becomes appsettings.json as JSON instead of XML. | |
| UML: (Unified Modeling Language) is a standardized general-purpose modeling language in the field of object-oriented software engineering .It is a visual language for specifying, constructing, and documenting the artifacts of systems. | **String interpolation**: $”this is {varName}” |
| INFO: Constructor shouldn’t have a return type. |  |
| Difference between I++ and ++I : ++I will return the new value | **padRight and padLeft:** are used to add characters to the beginning and end of your text |
| One of the benefits of UnitOfWork is that you can savechanges of more than one entity at the same time (like a transaction), creating a single database context class shared by all repositories.  Unit of Work in Repository Pattern: (increase the level of abstraction) is referred to as a single transaction that involves multiple operations of insert/update/delete and so on. Unit of Work is the concept related to the effective implementation of the Repository Pattern. For example: Insteed of making every repository create instance of DBContext we can create only one instance for all of these repositories as a one Unit Of Work:  public class UnitOfWork  {  private OrderRepository \_orderRepository;  private CustomerRepository \_customerRepository;  private Entities \_entities;  public UnitOfWork(Entities entities)  {  \_entities = entities;  }  public CustomerRepository CustomerRepo { get {  if(\_customerRepository == null)  \_customerRepository = new CustomerRepository(\_entities)  return \_customerRepository;  }  }  public OrderRepository OrderRepo  {  get  {  if (\_orderRepository == null)  \_orderRepository = new OrderRepository(\_entities)  return \_orderRepository;  }  }  public void Save() { \_entities.SaveChanges(); }  } | |
| $$ Gang Of Four GOF Design Patterns: using OOP to create a reusable designs and enhance the interaction between objects for the purpose of decupling, its categorized to 3 groups:  Creational Patterns: create objects in such a way that they can be decoupled from their implementing system.   * Factory Method: Create instances of derived classes, and let the subclass decides which class to instantiate   class ConcreteCreator : Creator  {  public override Product FactoryMethod(string type) {  switch (type) {  case "A": return new ConcreteProductA();  case "B": return new ConcreteProductB();  default: throw new ArgumentException("Invalid type", "type");  }  }  }   * Abstract Factory: create instances of several classes belonging to different families, so *Factory Method is a small part of Abstract Factory*   class HeroFactory : VehicleFactory {  public Bike GetBike(string Bike) {  switch (Bike) {  case "Sports":  return new SportsBike();  case "Regular":  return new RegularBike();  default:  throw new ApplicationException(string.Format("Vehicle '{0}' cannot be created", Bike));  }  }  public Scooter GetScooter(string Scooter) {  switch (Scooter) {  case "Sports":  return new Scooty();  case "Regular":  return new RegularScooter();  default:  throw new ApplicationException(string.Format("Vehicle '{0}' cannot be created", Scooter));  }  }  }   * Singleton: Ensures that a class can has only one instance (*lazy initialization*)   public class Singleton {  private static Singleton instance = null;  private Singleton() { }  public static Singleton GetInstance {  get {  return instance == null ? new Singleton() : instance;  }  }  }   * $ Builder: * $ Prototype:   Structural Patterns: identifying a simple way to realize relationships between entities.   * $ Addapter: * $ Bridge: * $ Composite: * $ Decorator: extends the functionality of individual objects by wrapping them with one or more decorator classes. These decorators can modify existing members and add new methods and properties at run-time. (*like using of Data Annotations*) * $ [Façade](https://www.dotnettricks.com/learn/designpatterns/facade-design-pattern-dotnet): * $ FlyWeight: minimize resource usage when working with very large numbers of objects. * $ Proxy: can be used when we don’t want to access to the resource or subject directly.   static class Program {  static void Main() {  var proxy = new Proxy();  proxy.Operation();  }  }  public abstract class SubjectBase {  public abstract void Operation();  }  public class RealSubject : SubjectBase {  public override void Operation() {  Console.WriteLine("RealSubject.Operation");  }  }  public class Proxy : SubjectBase {  private RealSubject \_realSubject;  public override void Operation() {  \_realSubject = \_realSubject == null ? new RealSubject() : realSubject;  \_realSubject.Operation();  }  }  Behavioural Patterns: how objects do interactions between each others.   * $ [Chain of Responsibility](https://www.dotnettricks.com/learn/designpatterns/chain-of-responsibility-design-pattern-dotnet): Passes a request among a list or chain of objects. * $ Command: * $ Interpreter: * $ Iterator: Provides a way to access the elements of a collection object in sequential manner without knowing its underlying structure. * $ Mediator: * $ Memento: * $ Observer: Allows an object (subject) to publish changes to its state and other objects (observer) that depend upon that object are automatically notified of any changes to the subject's state. * $ State: * $ Strategy: * $ Visitor: * $ Templete Method: used to define the basic steps of an algorithm and allow the implementation of the individual steps to be changed.   public abstract class AbstractClass{  public void TemplateMethod() {  Step1();  Step2();  Step3();  }  public abstract void Step1();  public abstract void Step2();  public abstract void Step3();  }  public class ConcreteClassA : AbstractClass {  public override void Step1() {  Console.WriteLine("Concrete Class A, Step 1");  }  public override void Step2() {  Console.WriteLine("Concrete Class A, Step 2");  }  public override void Step3() {  Console.WriteLine("Concrete Class A, Step 3");  }  } | |
| Difference between break and return: The return instruction is used either to return a function value or to terminate the execution of a function. ... The break instruction terminates the execution of the loop. | **C# Access Modifiers:**  **Public:** accessible from everywhere in our project.  **Private:** accessible only inside a class or a structure.  **Protected:** implies that the object is accessible inside the class and in all classes that derive from that class.  **Internal:** keyword specifies that the object is accessible only inside its own assembly  **protected internal**: we can access the protected internal member only in the same assembly or in a derived class in other assemblies (projects) |
| Inheritance: Parent = new Child() | **Difference between extend and implement:** extend is inheritance but implement is implementing an interface |
| Yeild return: The yield keyword signals to the compiler that the method in which it appears is an iterator block so it will customize iteration through a collection without creating a temporary collection so it should return IEnumerable collection | **Generics**: path parameters dynamically instead of using overload? |
| Difference between Boxing and Unboxing: Boxing is the conversion of the value type to an object type whereas, on other hands, the term Unboxing refers to the conversion of the object type to the value type. | **$ Implementing base type of generic type:** |
| Threading: allows concurrent programming of two or more parts of the program for maximizing the utilization of the CPU. Each part of a program is called Thread. So, in other words, threads are lightweight processes within a process. | |
| MultiThreading: Multithreading in C# is a process in which multiple threads work simultaneously. It is a process to achieve multitasking. It saves time because multiple tasks are being executed at a time. To create multithreaded application in C#, we need to use System.Threding namespace. | |
| Deadlock in C#: is a situation where two or more threads are frozen in their execution because they are waiting for each other to finish. | |
| Difference between Thread and Process and Task: A task can have multiple processes happening at the same time. And the process can have at least one thread.  *Task is preferred that Thread because when you run code using Task you make the ThreadPool create a new thread whenever a thread is available but when you run code using Thread you immediately create a new thread however may be at this time there no available threads* | |
| $ How to make a specific task\process excuted on a specific CPU: | |
| INFO: interface can’t be implemented by abstract class |  |
| $ ESP Enterprise Service Path: |  |
| Difference between Parse and Convert: Convert handle null values | **How system handle function calls:** by call stack |
| What Aggregate method do: is a function and is required to group together the values of the multiple rows as the input and returns the output as single value | **Difference between throw, throw(ex) and throws:**  - throw(ex) will reset your stack trace so error will appear from the line where throw(ex) written while throw does not reset stack trace and you will get information about original exception.  - throws used in class declaration |
| What is Enumerators: is a value data type in C#. It is mainly used to assign the names or string values to integral constants, that make a program easy to read and maintain. | |
| $ Bitwise AND (&): $ Bitwise Inclusive OR (|):  $ Bitwise Exclusive OR (XOR): | **String manipulation**: is the string operations that handle those such things can be Length property, ToCharArray(), ToLower(), ToUpper(), IndexOf(), LastIndexOf(), SubString(), CompareTo(), Trim(), TrimEnd(), TrimStart(), StartsWith(), EndsWith(), Split(), Insert(), Remove(), and Replace(). |
| Destructor: like constructor but method name preceded with (~) | **Modulus operator: %** |
| IDisposable interface: is an interface that contains a single method, Dispose(), for releasing unmanaged resources, like files, streams, database connections and so on. | |
| What is ViewState in .NET? preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back.  *You can enable\disable viewstate on the whole page of a particular element* | |
| In which event of page cycle is the ViewState available?  After the Init() and before the Page\_Load() | |
| What is the difference between Server.Transfer and Response.Redirect?  In Server.Transfer page processing transfers from one page to the other page without making a round-trip back to the client's browser. This provides a faster response with a little less overhead on the server. The clients url history list or current url Server does not update in case of Server.Transfer. | |
| INFO: Parent p = new Child() |  |
| INFO: if we have class Base and class Child, when we do Child c = new Child(); THEN *Base constructor run first then child constructor* | |
| INFO: We have to understand Data Structures well because the data structure object you decide to use will affect your code performance | |
| INFO: To deside what kind of data structures you should use, you need to consider these factors: size of data, is the data fixed or dynamic, how much frequent operations done (cost of operations), | |
| Data Structures has 2 types: Linear Data Structures like Array, Stack, Queue, Linked List and Hierachical Data Structures like Tree and Graphs. | |
| Keep in mind that, Recursive functions is used in most of Data Structures implementation.  Recursive Function: a function that call itself, example: Factorial & Fibunacci numbers implementation  public int Factorial(int n)  {  if (n >= 1) { return n \* Factorial(n - 1); }  else { return 1; }  }  public int Fibonacci(int n)  {  if (n >= 3) { return Fibonacci(n - 1) + Fibonacci(n + 2); }  else { return 1; }  } | |
| Big O is simply used to describe the performance of an algorithm, which describe the Time & Space complexity.  Big O: is how much the runtime of this code increases as the size of the input increases  - Constant O(1): the time doesnt increase by incresing input size.  - Linear O(n): the time consumed increase as much as the input increase.  - Quadratic O(n^2): | |
| Code Time Complexity: > Big O notation, the time complexity measure unit which describes performance which helps us to decide what resource like Hard Disk, RAM or Processor capacity should be used to run the application. O(1) Constant: static input  O(log(n)) Logarithmic: finding element in a sorted array with binary search (Binary search in C# Binary search works on a sorted array. The value is compared with the middle element of the array. If equality is not found, then the half part is eliminated in which the value is not there.)  O(n) Linear: finding max element in unsorted array  O(n log n) Linearithmic: sorting elements in array with merge sort  O(n2) Quadratic: sorting array with bubble sort  O(n3) Cubic:  O(2n) Exponential: find all subsets  O(n!) Factorial: | |
| How to calculate the Big O: (ex: cn + dn^2 + e)  - choose the fastest growing term > dn^2  - Remove the coeffecient > n^2 > Quadratic  - so in general, you can check every line of code, how this line can be affected according to the input size. | |
| INFO: Code performance is always to Save Time and Save Space, so you need to decide which factor is more important so if you are developing a mobile app then saving space is much important and so on. | |
| *Data Structures*  Array: list of a specific count of elements with a specific data type,  Array is a collection of items that should be with a fixed length, and we can't add more numbers to exceed the length BECAUSE, in memory we book a specific number of bytes to insert array values sequentialy so that we may have other variables book the next slots of memory so we can't insert new values beside the old values in memory, because to insert array values we need a bunch of consective bytes.  and what about the List (the dynamic array) its working like everytime you add a new value it defines a new array with the length of old + new values and so on  Time needed to access an element in array is O(1) constant time because the array elements are stored in a contigious block of memory  List: is a collection of objects with the same data type, it has no good memory utilization since it creates a new memory locations for the whole elements every time you add or remove elements.  ArrayList: list of elements with dynamic data type  LinkedList: every node has 2 locations in memory, one for the node value and another for a pointer to the next node,  to access element in the linked list, it takes O(n),  Apart from storing linked list elements into memory, we need also another memory slot to save the address of the first element in the linked list (a pointer to head) which called the identity of the linked list.  what is distinguish linked list than array is that if you want to add\remove an element to\from the linked list you will just need to reference the next element address in its value and also set the new element's address as reference in the previous element, but in array you will need to shift the index increase\decrease for all elements after\before the element you want to delete\add.  So in general linked list need extra memroy for every element to store a reference to the next element, but array need only one slot of memory for every element.  public class LinkedTest<T> where T : class  {  public int ListCount(NodeTest<T> head)  {  int count = 0;  NodeTest<T> currentNode = head;  while (currentNode != null)  {  currentNode = currentNode.Next;  count++;  }  return count;  }  }  public class NodeTest<T> where T : class  {  public int Value { get; set; }  public NodeTest<T> Next { get; set; }  }  Doubly Linked List: is a normal linked list but each element has 2 references, one for the next node and another for the previous node.  Dictionary: list of key, value pairs for a specific key value data types, it talk O(1) time to find an element  Hashtable, HashMap: like Dictionary list of key, value pairs for a dynamic key value types, Hash Table: a collection of key-and-value pairs that are organized based on the hash code of the key, which uses the Hash Function to hash the key  a methodology for storing array elements in indexes related to its values (do some equations to calculate the element index from its value)  HashSet: list of distinct elements (it ignore duplicates)  HashMap data structure: like Dictionary (key & value) but with a dynamic data types  Stack: LIFO last in first out  Queue: FIFO first in first out  Most common examples of using Stack data structure is: Function Calls so the last function enter stack is the first one to finish, Also one more example is the Undo function to reverse changes one by one most recently.  - Stack and Queue can be implemented using Array or LinkedList  Tree data structure is used to store a hierarchical data, where each node has 1 or more childs  Binary Tree: is a normal tree where each node has only 2 childs  Binary Search Tree (BST): is a binary tree where for every node, all nodes on its right side has value greated than its value and all nodes to the left has value less than its value,, exactly like the Binary Search in general we should have the array sorted so this is the same we assume the tree as a binary search tree if each node in the left is less than its sibling in the right.  so finding an element in a binary search tree is just costing O(log n).  so if you want to find the minimum value in a BST, you just go to the left as much as it exists (the last element), and the miximum is the last element in the right.  Hight of a node: is the number of edges in the longest path from the node to the leaf node.  Graph: is a Tree but with no leaves, where the number of nodes equal the number of edges.  Directed Graph: where the edges between nodes are in one direction, but the Undirected Graph, the edges have 2 directions.  INFO: social networks are a good example for the UnDirected Graph where each node has 2 directions to each other. | |
| Popular algorithms you should know:  Linear Search: It sequentially checks each element of the list until a match is found or the whole list has been searched, it cost O(n)  Binary Search: assuming that the elements are sorted, go to the middle point of array and check if the target is greater than the middle then move to the right and repeat:  Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item  public static int BinarySearch(int[] source, int target)  {  int left = 0, middle = 0;  int right = source.Length - 1;  while (left < right)  {  middle = (left + right) / 2;  if (source[middle] == target)  return middle;  else if (target < source[middle])  right = middle - 1;  else if (target > source[middle])  left = middle + 1;  }  return middle;  }  DFS Depth First Search: is used to find an element in a graph\tree by visiting all graph nodes and exploring all connected nodes as following, it cost O(n):  1. visit any node as a start node, and push to STACK.  2. loop through the stack:  - for current node, explore one of its connected node and check if its not exist in the stack (not visited) then push to the stack otherwise pop from the stack untill you reach this visited node and check the other connected node for this visited node.  - repeat the process until the stack become empty  BFS Breadth First Search: is used to find an element in a graph\tree by visiting all graph nodes and exploring all connected nodes as following, it cost O(n):  1. visit any node as a start node, and push to QUEUE.  2. loop through the queue  - for current node, explore all connected nodes by enqueueing them into queue and delete this node  - repeat untill the queue is empty  Quick Sort: its a devide and conquer algorithm where every element find its sorted position that follow the below steps:  - partition list into 2 sub-arrays by selecting the pivot element:  - consider pivot as the first element.  - loop through elements if found element < pivot then swap it, until the left and right pointers replaced  - swap the pivot with the last swaped location location (move pivot to its sorted location).  - return the last swaped location as the new pivot  - repeat the quickSort again from left to pivot  - repeat the quickSort again from pivot to right  #region Quick Sort  public static int[] QuickSort(int[] given\_array, int start, int end)  {  if (start < end)  {  int pivotIndex = QuickSortPartition(given\_array, start, end);  QuickSort(given\_array, start, pivotIndex);  QuickSort(given\_array, pivotIndex + 1, end);  }  return given\_array;  }  public static int QuickSortPartition(int[] given\_array, int left, int right)  {  int pivot = given\_array[left];  int swapIndex = left;  for (int i = left + 1; i < right; i++)  {  if (given\_array[i] < pivot)  {  swapIndex++;  SwapArray(ref given\_array, i, swapIndex); // sort these 2 elements  }  }  SwapArray(ref given\_array, left, swapIndex); // move pivot to its sorted location  return swapIndex;  }  public static void SwapArray(ref int[] given\_array, int firstIndex, int secondIndex)  {  int temp = given\_array[firstIndex];  given\_array[firstIndex] = given\_array[secondIndex];  given\_array[secondIndex] = temp;  }  #endregion  Merge Sort: based on the Devide and Conquer approach, we devide the list into sub lists until we have a very small lists (1 element) then merge it up (sorted) one by one until merge it all again but in a sorted order, it cost O(n log n)  #region Merge Sort  public static void Merge(int[] arr, int left, int middle, int right)  {  int leftArrayLength = middle - left + 1;  int rightArrayLength = right - middle;  // Devide array  int[] leftArray = new int[leftArrayLength], rightArray = new int[rightArrayLength];  for (int x = 0; x < leftArrayLength; x++)  leftArray[x] = arr[left + x];  for (int y = 0; y < rightArrayLength; y++)  rightArray[y] = arr[middle + 1 + y];  // Maintain current index of sub-arrays and main array  int i, j, k;  i = 0;  j = 0;  k = left;  // Until we reach either end of either L or M, pick larger among  // elements L and M and place them in the correct position at A[p..r]  while (i < leftArrayLength && j < rightArrayLength)  {  if (leftArray[i] <= rightArray[j])  {  arr[k] = leftArray[i];  i++;  }  else  {  arr[k] = rightArray[j];  j++;  }  k++;  }  // When we run out of elements in either L or M,  // pick up the remaining elements and put in A[p..r]  while (i < leftArrayLength)  {  arr[k] = leftArray[i];  i++;  k++;  }  while (j < rightArrayLength)  {  arr[k] = rightArray[j];  j++;  k++;  }  }  public static int[] MergeSort(int[] given\_array, int left, int right)  {  if (left >= right)  {  return given\_array;  }  int middle = left + (right - left) / 2;  MergeSort(given\_array, left, middle);  MergeSort(given\_array, middle + 1, right);  Merge(given\_array, left, middle, right); // merge two subarrays L and M into arr  return given\_array;  }  #endregion  Bubble Sort: loop twice in list elements, so it cost O(n^2)  - loop to compare each element with the next one, if the next one is greater, then swap  - execlude the last element from the list because it will be already sorted  - repeat first step  - exclude the last 2 elements, because it will be sorted  - repeat this until either excluding all elements or found an iteration without any replacement  #region Bubble Sort  public static int[] BubbleSort(int[] given\_array)  {  int temp = 0;  bool alreadySorted = true;  for (int i = 0; i < given\_array.Length; i++)  {  alreadySorted = true;  for (int j = 0; j < given\_array.Length - 1; j++)  {  if (given\_array[j] > given\_array[j + 1])  {  temp = given\_array[j + 1];  given\_array[j + 1] = given\_array[j];  given\_array[j] = temp;  alreadySorted = false;  }  }  if (alreadySorted) { break; /\*early escape\*/ }  }  return given\_array;  }  #endregion  Selection Sort: simply is foreach element we select the elment that is less than it and swap them, so it cost O(n^2)  #region Selection Sort  public static int[] SelectionSort(int[] given\_array)  {  int arrayLength = given\_array.Length;  // One by one move boundary of unsorted subarray  for (int i = 0; i < arrayLength - 1; i++)  {  // Find the minimum element in unsorted array  int min\_idx = i;  for (int j = i + 1; j < arrayLength; j++)  if (given\_array[j] < given\_array[min\_idx])  min\_idx = j;  // Swap the found minimum element with the first element  int temp = given\_array[min\_idx];  given\_array[min\_idx] = given\_array[i];  given\_array[i] = temp;  }  return given\_array;  }  #endregion  Insertion Sort: is simply foreach element we insert it in its sorted position, so it cost O(n^2)  #region Insertion Sort  public static int[] InsertionSort(int[] given\_array)  {  int n = given\_array.Length;  for (int i = 1; i < n; ++i)  {  int element = given\_array[i];  int j = i - 1;  // Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current position  while (j >= 0 && given\_array[j] > element)  {  given\_array[j + 1] = given\_array[j];  j = j - 1;  }  given\_array[j + 1] = element;  }  return given\_array;  }  #endregion  Prime number: is the numbers that has zero reminder % in only 1 and itself (accept devide by 1 and itself)  private static bool IsPrimeNumber(int number)  {  int m = number / 2;  bool isPrime = true;  for (int i = 2; i <= m; i++)  {  if (number % i == 0)  {  isPrime = false;  break;  }  }  return isPrime;  }  Factorial:  Fibonacci numbers: Is a list of numbers where every number is the sum of the two preceding ones. | |
| Difference between Float, Double and Decimal: main difference is the size of fraction  float 7 digits of precision  double has about 15 digits of precision  decimal has about 28 digits of precision | |
| Difference between Parce and ParceExact: both converts string to date but ParseExact enable parsing to a particular format. | |
| GIT Steps: Stage, Commit, and Push to pull request that you previously created. > Git is a distributed version control system but TFS is a centralized VCS, so in GIT, every one has a local copy of the source code and also support branching.  > Git Command (git config) has a 3 levels of configuration: (git config --global) which use the global config file to be applied on all users that use this computer, (git config --system) which use the system config file to be applied on all repositories for current user and (git config --local) use use the repository config file to be applied on a particular repository  > Every Git Commit contains a complete snapshot of the respo.  > Use command (git status) to show the status of your changes, or (git status -s) for short status  > Git > local changes > stagging are (local) the stagging are is called (index) > push to remote server  > If you want to review the stagged files before commit, use this command (git diff --staged)  > To view history of repo activities, use (git log)  > The branch is a new commit that pointing to some particular parent (commit in another branch)  > The merge is nothing but a new commit that pointing to 2 parents (2 commits in 2 branches)  > (git rebase) is simply making the branch pointing to some particular parent branch(commit) | |
| Reflection: is used to retrieve metadata on types at runtime. ... In using reflection, you get objects of the type "Type" that can be used to represent assemblies, types, or modules.  You can use reflection to create an instance of a type dynamically and even invoke methods of the type. | |
| $ Disadvantages of reflection: Performance | |
| C# object indexer: An indexer is a special type of property that allows a class or a structure to be accessed like an array for its internal collection. C# allows us to define custom indexers, generic indexers, and also overload indexers, An indexer can be defined the same way as property with this keyword and square brackets []  class StringDataStore  {  private string[] strArr = new string[10]; // internal data storage  public string this[int index]  {  get => strArr[index];  set => strArr[index] = value;  }  } | |
| Difference between Property and Field:  Property is a member that provides a flexible mechanism to read, write or compute the data of a private field.  Field is the private members of a class that contain values specific to the object  // this is a field. It is private to your class and stores the actual data.  private string \_myField;  // this is a property. When accessed it uses the underlying field,  // but only exposes the contract, which will not be affected by the underlying field  public string MyProperty  {  get {  return \_myField;  }  set {  \_myField = value;  }  } | |
| SOLID Principles : an acronym for:  S (Single Responsibility Principle): class should have responsibility for just a single piece of that program's functionality, it’s not preferred to handle exception inside business class but do it in an external class Exception Handler so this class will have a single responsibility.  O (Open Close Principle): states "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification", it’s not preferred to edit class code but add an extension with the new code.  The extension here means to implement an extension method or create a new child class, suppose that you have a class (Shape) that has a (CalcArea) method to calculte shape area, this method has a (if shape is circle then do bla bla bla, else id square then do bla bla) and now you want to add more shape to calculate rectangle are then the principle here is not change that class but idealy to create an abstract class (Shape) that has also an abstract method (CalculteArea) and create a class for eac shape to do special implementation for CalculateArea method  L (Liskov Substitution Principle): the subtype must be substitutable for its base type, so suppose that you have a Rectangle class which has a SetWidth method, and now you want to implement a Square class then you should NOT inherit from Rectangle class to reuse the SetWidth method but idealy to create a base class for both Shape class, it’s not preferred that the child class can’t use all objects of the parent class so you better create a bridge class: public class Bird{  }  public class FlyingBirds : Bird{  public void fly(){}  }  public class Duck : FlyingBirds{}  public class Ostrich : Bird{}  so FlyingBirds here is the bridge class  I (interface-segregation principle): (Keep your interface as small as possible) the child class should not implement not related methods of big interface but idealy you should break down the big interface into a small interfaces based on the business, it’s not preferred to add more methods to interface but create a new one in which you add the new needed methods.  D (Dependency inversion principle): (Depend upon abstractions\interfaces rather than concrete class). High level modules should not depend on low level modules but both should depend on abstractions and those abstractions should not depend on details.  It’s not preferred to do this:  class Customer : IDiscount, IDatabase  {  private IException obj;  public virtual void Add(int Exhandle) {  try {  // Database code goes here  }  catch (Exception ex) {  if (Exhandle == 1) {  obj = new MyException();  }  else {  obj = new EmailException();  }  obj.Handle(ex.Message.ToString());  }  }  But instead do this:  class Customer : IDiscount, IDatabase  {  private Ilogger obj;  public Customer(ILogger i)  {  obj = i;  }  }  IDatabase i = new Customer(new EmailLogger()); | |
| Difference between Dependency Inversion and Dependency Injection: dependency inversion is the principle but dependency injection is a way of applying this principle. | |
| What is dependency injection: is a technique whereby one object (or static method) supplies the dependencies of another object, to prevent the high level modules being depends on the low level modules,  *So this is a way of applying Dependency Inversion principles of SOLID* | |
| Pass by reference: When you pass a reference type variable from one method to another, it doesn't create a new copy; instead, it passes the address of the variable. If we now change the value of the variable in a method, it will also be reflected in the calling method.  Pass by value: When you pass a value type variable from one method to another method, the system creates a separate copy of a variable in another method, so that if value got changed in the one method won't affect on the variable in another method. | |
| Caching Server Side: Caching is a technique where we can store frequently used data Session\User Caching: data is stored at the user level Session["userName"] = "Omar";  Application Caching: data stored over the application level HttpContext.Current.Cache.Insert(args); Memory Caching: used for when you want to implement cache in a single process. When the process dies, the cache dies with it. (namespace > System.Runtime.Caching) var cache = MemoryCache.Default; var item = cache.Get("CacheKey");  Disk Caching: usually used for caching files on disk  Distrbuted Caching: A distributed cache is a cache shared by multiple app servers, typically maintained as an external service to the app servers that access it. A distributed cache can improve the performance and scalability of an ASP.NET Core app, especially when the app is hosted by a cloud service or a server farm.  Redis distributed caching: [Redis](https://redis.io/) is an open source in-memory data store, which is often used as a distributed cache. You can use Redis locally (using Docker Desktop app), and you can configure an [Azure Redis Cache](https://azure.microsoft.com/services/cache/) for an Azure-hosted ASP.NET Core app. | |
| Data Driven Development: is something that one can make changes to the logic of the program by editing not the code but the data structure. The application has nothing but rules (Engine) . What makes it useful is the data that will flow through it. | |
| Unit Testing: is to validate that each unit of the software code performs as expected Test Driven Development (TDD): relying on a testing framework (MS Test, NUnit or XUnit), and writing unit tests before coding to be automated lated | |
| Domain Driven Development: Simply, the domain is an area of knowledge, so DDD is the concept that the structure and language of software code should match the business domain. | |
| $ Most popular design patterns:  Dependency Injection: injecting the object of a class into a class that depends on that object. Singleton: is a class that only allows a single instance of itself to be created and usually gives simple access to that instance.  Observer: is a behavioral design pattern that allows one objects to notify other objects about changes in their state. $ the design pattern which structure app to change without modifying?? | |
| $ How to execute a transaction using ADO.NET and EF: | |
| Length of memory allocated for Int data type? 4 bytes | |
| $$ If you have a List<int>, Where should it be stored? and If you have 1 million item in this list where do it stored and are we have a limitations in memory? | |
| $$ How dictionary data type works? How many operations does .net talks to search for an item inside dictionary? and what is the complexity? | |
| $$ If you have a memory leackage in IIS, How would you troubleshoot this problem? | |
|  | |
|  | |

|  |  |
| --- | --- |
| Entity Framework | |
| ORM: (Object Relational Mapping) is a technique that lets you query and manipulate data from a database using an object-oriented paradigm. | |
| Difference between IEnumerable and IQuerable in EF: IEnumerable execute select query on server side, load data in-memory on client side and then filter data while IQueryable execute select query on server side with all filters. | **Difference between FirstOrDefault and First in EF:** First() will throw an exception if there is no result data for the supplied criteria, whereas FirstOrDefault() returns a default value (null) if there is no result data. |
| Difference between SingleOrDefault and Single in EF: SingleOrDefault() returns default value of the data type of a collection if a collection is empty, includes more than one element or finds no element or more than one element for the specified condition. | **INFO**: When write a LINQ query ended with .ToList() > it will return IEnumerable. |
| Write a left\right\inner join query with LINQ:  Add this right after join from z in jointable.DefaultIfEmpty() | |
| INFO: Entity Framework is injection safe since it always generates parameterized SQL commands which help to protect our database against SQL Injection. | |
| DbSet is a typed entity set which is used to perform create, read, update, and delete operations on a particular entity. DbSet is can only be created from a DbContext instance. DbSet does not support the Entity SQL methods. | |
| How to execute plain SQL in EF6?  DbSet.SqlQuery()  DbContext.Database.SqlQuery()  DbContext.Database.ExecuteSqlCommand() | |
| How does EF support Transaction?  In EF, whenever you execute SaveChanges() to insert, update or delete data into the database, it wraps that operation in a transaction. So, you don’t need to open a transaction scope explicitly. | |
| What is the purpose of a DBContext class?  DbContext as the database connection and a set of tables, and DbSet as a representation of the tables themselves. The DbContext allows you to link your model properties (presumably using the Entity Framework) to your database with a connection string. | |
| INFO: EF uses the Disconnected mode while connecting to DB. | |
| EF types of loading:  Lazy Loading: default behavior which retrieves all related entities, it can be disabled by adding the below in the constructor of Entities class this.Configuration.LazyLoadingEnabled = false;  Eager Loading: entity.Include(x => x.SubEntity)  Explicit Loading: Use the Load() method to load related entities explicitly.  Context.Entry(entityObject).Reference(x => x.subEntity).Load(); | |

|  |  |
| --- | --- |
| MVC | |
| MVC Life Cycle: Request > Controller > Action Excution > Result Excution > View Engine > Result Excution > Response | * **MVC Filters types and order of executing**:  Authorization filters * Action filters * Result filters * Exception filters |
| Difference between ViewBag, ViewData and TempData:  ViewBag and ViewData are used to pass the data from Controller action to View and TempData is used to pass the data from action to another action or one Controller to another Controller. | **What exactly the use of ViewModel in MVC:** to encapsulate the multiple entities into single entity. It is basically a combination of data models into single object and rendering by the view. Sometimes it is required to show the multiple entities data on view which is coming from different data model classes. |
| What is MVVM: is an architectural pattern that separate the development of the graphical user interface from the development of the business logic or back-end logic so that the view is not dependent on any specific model platform. | |
| difference between Temp data, View, and View Bag?  Temp data: It helps to maintain data when you shift from one controller to other controller.  View data: It helps to maintain data when you move from controller to view  View Bag: It's a dynamic wrapper around view data | |
| Ordered MVC Filters:  Authorization filters  Action filters  Response filters  Exception filters | |

|  |  |
| --- | --- |
| SQL | |
| SQL Server Integration Services (SSIS): is a component of the Microsoft SQL Server database software that can be used to perform a broad range of data integration and data transformation tasks.  It is basically an ETL (Extract, Transform and Load) tool part of the Microsoft Business Intelligence suit that is mainly used to achieve Data Integration.  SQL Server Reporting Services (SSRS): is a solution that customers deploy on their own premises for creating, publishing, and managing reports, then delivering them to the right users in different ways, whether that’s viewing them in web browser, on their mobile device, or as an email in their in-box. SSRS is a reporting software that allows you to produce formatted reports with tables in the form of data, graph, images, and charts. These reports are hosted on a server that can be executed any time using parameters defined by the users. It is part of Microsoft SQL Server Services suite.  You first create the report itself (.rdl) by desiging it and provide the data source (db connection) and the data set (query or stored procedure) and then configure the subscription, all of this after configuring the SSRS reporting server and configuring the mail settings, from the SQL Server you can connect to thr Reporting Server and click server properities to get the URL and paste this URL in VS > project properities > Target ServerURL and then right click on project and choose deploy (dont forget to Run As Admin), to view the report open the Web Portal url in IE with Run As Admin | |
| Conceptual and Logical data model: A conceptual data model identifies the highest-level relationships between the different entities. ... A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. | |
| Computed Column: is writing equation in (Computed column specification) in column properties and its usually a combination of a couble of columns.  And if (Persisted) property is set to off then the calculated column will be just a virtual column. No data for this column will be stored on disk and values will be calculated every time when referenced in a script. | |
| DbUp: is a . NET library that helps you to deploy changes to SQL Server databases. It tracks which SQL scripts have been run already, and runs the change scripts that are needed to get your database up to date. | |
| INFO: in CHARINDEX() function the first index in the string is 1 not 0 :D | **Difference between Unique key and Primary key:** Primary Key is *not allowing NULL* and used to identify a row (record) in a table, whereas Unique-key is to prevent duplicate values in a column |
| $ SQL Table, Column standards: | **$$ Extranet Views:** |
| Difference between Procedure and Function: The function must return a value but in Stored Procedure it is optional. Even a procedure can return zero or n values. Functions can have only input parameters for it whereas Procedures can have input or output parameters. | |
| $ Is Indexing affect update, delete queries performance? | |
| $ Creating many indexes will increase or decrease performance? | |
| $$ Aside | |
| SQL difference between:  Char: store string in a fixed length (static memory allocation) Varchar: store string in a variable length (dynmic memory allocation) and stores data at 1 byte per character. NChar\NVarchar: N refers to national character so the only difference is that supporting national characters and stores data at 2 bytes per character. | |
| Deference between Truncate and Delete: - Truncate reseeds identity values, whereas delete doesn't.  - Truncate removes all records and doesn't fire triggers.  - Truncate is faster compared to delete as it makes less use of the transaction log.  - Truncate is not possible when a table is referenced by a Foreign Key or tables are used in replication or with indexed views. | **What is View:** is a virtual table base as a result set of a stored query on the data. |
| Command types in SQL: Data Definition Language (DDL): make/perform changes on the physical structure of any table residing inside a database like Create, Alter, Rename, Drop and Truncate Data Manipulation Language (DML): changing and manipulation inside those tables and databases is done using DML commands like Insert, Delete, Update, Lock and Merge  Data Query Language(DQL): fetch data from database/tables like Select Data Control Language(DCL): provides special privilege access to users and is also used to specify the roles of users accordingly like Grant and Revoke  Transaction Control Language(TCL): used to rollback or commit the changes in the database like Roll Back and Commit | |
| What is the use of OrderBy, GroupBy:  ORDER BY is used to sort a result by a list of columns or expressions. GROUP BY is used to create unique combinations of a list of columns that can be used to form summaries. | **What is distinct:** The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values. |
| SQL Cursor to loop on records:  DECLARE @MyCursor CURSOR;  DECLARE @MyField YourFieldDataType;  BEGIN  SET @MyCursor = CURSOR FOR  select top 1000 YourField from dbo.table  where StatusID = 7  OPEN @MyCursor  FETCH NEXT FROM @MyCursor  INTO @MyField  WHILE @@FETCH\_STATUS = 0  BEGIN  /\*  YOUR ALGORITHM GOES HERE  \*/  FETCH NEXT FROM @MyCursor  INTO @MyField  END;  CLOSE @MyCursor ;  DEALLOCATE @MyCursor;  END; | |
| Explain joins:  (INNER) JOIN: Returns records that have matching values in both tables  LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table  RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table  FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table | |
| Cross join: Is returning both tables select \* from employee cross join department > is the same as > select \* from employee, department | |
| Union & Union All: both used to combines the result set of two or more SELECT statements. *Union will return the distinct rows but Union All will return the duplicated ones* | |
| Having Clause:  SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5 ORDER BY COUNT(CustomerID) DESC; | |
| System-Versioned Temporal Table: is a type of user table designed to keep a full history of data changes to allow easy point in time analysis. This type of temporal table is referred to as a system-versioned temporal table because the period of validity for each row is managed by the system (i.e. database engine).  CREATE TABLE dbo.Employee  (  [EmployeeID] int NOT NULL PRIMARY KEY CLUSTERED  , [Name] nvarchar(100) NOT NULL  , [Address] nvarchar(1024) NOT NULL  , [AnnualSalary] decimal (10,2) NOT NULL  , [ValidFrom] datetime2 GENERATED ALWAYS AS ROW START  , [ValidTo] datetime2 GENERATED ALWAYS AS ROW END  , PERIOD FOR SYSTEM\_TIME (ValidFrom, ValidTo)  )  WITH (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory)); | |
| Difference between:  Scalar-valued: return a single value as a return value, not as a result set  Table-valued: returns your specified columns for rows in your table meeting your selection criteria.  Aggregate functions: returns a calculation across the rows of a table -- for example summing values. | |
| $$ DB Housekeeping tasks: | |
| SQL User-Defined Data Type: When there is a requirement to create a column with a datatype in the database and creating the same data type columns multiple times by multiple people.  create > CREATE TYPE TypeName FROM VARCHAR(100)  use > DECLARE @Name TypeName | |
| What is Temp Table: is a database table that exists temporarily on the database server. A temporary table stores a subset of data from a normal table for a certain period of time. ... Temporary tables are stored inside “tempdb” which is a system database. | **What is DB normalization:** is the process of structuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. |
| What is SQL Indexes: Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. Clustered Index: the rows are stored physically on the disk in the same order as the index. Therefore, there can be only one clustered index.  CREATE CLUSTERED INDEX IX\_tblStudent\_Gender\_Score  ON student(gender)  Or by creating a clustered index using a composite key  CREATE CLUSTERED INDEX IX\_tblStudent\_Gender\_Score  ON student(gender ASC, total\_score DESC)  Non-Clustered Index: there is a second list that has pointers to the physical rows. You can have many non clustered indices, although each new index will increase the time it takes to write new records.  create NonClustered index IX\_table\_name\_column\_name  on table\_name (column\_name ASC) – or more columns  *It is generally faster to read from a clustered index if you want to get back all the columns. You do not have to go first to the index and then to the table.* | |
| SQL Full text search: refers to the functionality in SQL Server that supports full-text queries against character-based data. These types of queries can include words and phrases as well as multiple forms of a word or phrase.  To support full-text queries, *full-text indexes* must be implemented on the columns referenced in the query. | **$** **How to avoid writing sub queries?** using Exists(select …) and Not Exists(Select …) and others |
| SQL Server Deadlock: is a special concurrency problem in which two or more transactions block the progress of each other. The first transaction has a lock on some database object that the other transaction wants to access, and vice versa. (may fire because an indexes on table)  *It always fire error code: 1205*  To resolve a deadlock, *SQL Server has to rollback the cheapest of the 2 transactions. In the context of SQL Server, the cheapest transaction (the victim) is the transaction that has written the fewer bytes to the transaction log.*  SQL Server implements the deadlock detection in a background process called the Deadlock Monitor.  *You can avoid deadlocks by creating indexes on table foreign keys* | |
| How to enhance SQL query performance?   1. Use joins instead of sub-queries. 2. Avoid \* in SELECT statement. Give the name of columns which you require. 3. Use EXISTS and NOT EXISTS instead of IN and NOT IN to check existence of data. 4. Use WITH (NOLOCK) while querying the data from any table. 5. Use WHERE expressions to limit the size of result tables that are created with joins. 6. Use SET NOCOUNT ON to prevent sql server from calculating the effected rows. 7. Use UNION ALL in place of UNION if possible. 8. Avoid Cursors since cursor are very slow in performance. 9. Use Schema name before SQL objects name. 10. Choose appropriate Data Type. E.g. To store strings use varchar in place of text data type. 11. Avoid nchar and nvarchar if possible since both the data types takes just double memory as char and varchar. 12. Use Indexes. 13. Most selective columns should be placed leftmost in the key of a non-clustered index. 14. Drop unused Indexes. 15. Better to create indexes on columns that have integer values instead of characters. Integer values use less overhead than character values. 16. Use Stored Procedure for frequently used data and more complex queries. | |
| What is SQL Profiler: a tool for tracing, recreating, and troubleshooting problems in MS SQL Server | |
| $$ What is the execution plan for sql query?  is a visual representation of the operations performed by the database engine in order to return the data required by your query. *Its used to monitor query execution in purpose of optimization*  *The most important thing to enhance the query is to reduce the Reads and Writes in memory*,,, It has 2 types:  Estimated Execution Plan & Actual Execution Plan. | |
| What is SQL Transaction? is a sequence of operations performed (using one or more SQL statements) on a database as a single logical unit of work. The effects of all the SQL statements in a transaction can be either all committed (applied to the database) or all rolled back (undone from the database) | |
| RTRIM & LTRIM: is to remove spaces from left and right | |
| Table partitioning is a way to divide a large table into smaller, more manageable parts without having to create separate tables for each part. Data in a partitioned table is physically stored in groups of rows called partitions and each partition can be accessed and maintained separately. Partition Column: Data in a partitioned table is partitioned based on a single column, the partition column, often called the partition key. Only one column can be used as the partition column, but it is possible to use a computed column. Partition Function defines how to partition data based on the partition column. | |
| Common Table Expression CTE, is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. The CTE can also be used in a View. ;with CTE\_Name(Col1, Col2) as  (  select database\_id Col1, name Col2  from sys.databases  )  select \* from CTE\_Name | |
| CTE can be refertenced only once, right after defining it, so if you want to use a permenant table consider using temp table or table variable | |
| Difference between CTE and Temp Table: CTE can only be used in the current query scope whereas a temporary table or table variable can exist for the entire duration of the session allowing you to perform many different DML operations against them. | |
| Difference between temp table and table variable: Temporary Tables are physically created in the tempdb database. These tables act as the normal table and also can have constraints, index like normal tables. Table Variable acts like a variable and exists for a particular batch of query execution. ... It is created in the memory database but may be pushed out to tempdb. | |

|  |  |
| --- | --- |
| Encryption | |
| What is Encryption: is to cypher data using a public key and decrypt it using the private key, the private key is usually accessed from the certificate which previously setuped on server. (RSA)  *To encrypt data you use something called a cipher algorithm.*  Single Key "AES": one key can be used to encrypt, and then decrypt the message. You need to securely share that key with the system you’re exchanging encrypted data with, otherwise other people can decrypt your data, or pretend to create encrypted data on your behalf.  Two Key "RSA”: a pair of keys is used, one you keep to yourself (private), and one you share with the rest of the world (public),  the public key to encrypt the data, which can only then be decrypted with the private key. | |
| Hashing: sender and receiver has the same hash key, so sender send the plaintext data plus the hashed version so receiver do hash the plaintext data as well and compare the hashed result with the hashed data received from the sender. Example:MD4 (deprecated) and SHA256. | |
| Encryption salting: adding random data to the input of a hash function to guarantee a unique output | **INFO**: Hashing: is signature approve.. “SHA” |
| Difference between Hashing and Encryption: is that hashing converts the data to message hash that identify the message so that when the receiver recieves your message he will hash the message and compared the generated hash with the one sent in the message, and it should be the same, so Hashing is one-way, while encryption uses a key to convert the message to transmit into an unrecognizable format, two-way encryption.  One of the most practical example for Hashing is when you want to check if some data changed in a big combination of data in database, you can simply prepare a selec qurery for it and hash the result and compare the hash with the last hashed value so if they are not the same then one or more columns\tables data changed. | |
| HTTPS Certificate: setuped on the server which contains some data about the site owner and the public and private keys used for encryption and decryption. | |
| $ How TLS works and relation to SSL: | |

|  |  |
| --- | --- |
| General Tips | |
| OWASP (Open Web Application Security Project). Talking about most critical security risks to web applications like (SQL Injection, XSS, CSRF, etc...). | **Data Reconciliation** (Verification of Data Migration). |
| MIGs Payment:  - Build your own form,  - Submit this form to your server with an encrypted data.  - Server decrypt data using his private key from certificate which previously setuped on server then do some operations like logs or whatever (prepayment operations) and get payment configuration data from DB (MIGS submit URL, MIGS return URL, Merchant ID, etc..) and build the MIGS formal form with a specific form id and specific form elements IDs then return back to client with a hashed version of data. - The client just auto-submit it to the MIGS submit URL.  - MIGS perform his operations and return with the status code to the MIGS return URL. (with hashed data), so in page load I hash and compare data and some logs\operations then tell the user about the result.  - Every 24 hours EOD , MIGS drop some files to my server using SFTP, which contains all successful transactions (as a verification)  - I have a Windows service to run periodically to read this files and mark these transactions as verified in DB (Reconciliation). | |
| $ What is Fiddler: | **$ brute force attack** |
| AppPool Start mode: OnDemand & AlwaysRunning | **$** **AppPool Idle Time-Out:** |
| INFO: AppPool "Service Unavailable" Response Type > HTTP & TCP > TCP is better | **$** **IIS MIME Types:** |
| Site Binding: enable ports and (http & https). | **$** **Patch file:** |

|  |  |
| --- | --- |
| Technical Task Acceptance Criteria | |
| Should apply all required points *letterly*. | Should be well tested. |
| Apply best practice with simplicity. | If Typescript then avoid using any as much as possible. |
| Error handling. | Apply Unit Testing. |
| Remove any unused code. | Should be well structured. |
| Should make sure that it works (Simulate the reviewer position) especially if uploaded on github or gitlab and don’t forget to set installation instruction in readme file. | |

**EPAM**

1. What do you know about EPAM Systems
2. OOP
3. $ SOLID Principles especially Liskov
4. Design Patterns
5. $ Unit Tests
6. Event Handlers
7. Memory Management
8. C# Types
9. Data Structures
10. $ Value Type and Reference Type
11. $ Pass by value and Pass by reference
12. $ Code Quality Factors:
13. $ Coding Standards:
14. $ Quality Gates:
15. CI/CD:
16. CI/CD Pipelines:
17. Regression Test:
18. Integration Test
19. Smoke Test
20. Advantages of ORM:
21. Disadvantages of ORM
22. $ SQL Profilers
23. How to enhance SQL Query
24. SQL Table, Column standards
25. Difference between .Net Framework and .Net Core
26. $ .Net Core Middleware
27. Disadvantages of reflection
28. $ Using operator
29. $ IDisposable interface
30. How to execute a transaction using ADO.NET
31. $ Indexing in SQL (Clustered & NonClustered)
32. Is Indexing affect update, delete queries performance?
33. Creating many indexes will increase or decrease performance?
34. Difference between Primary Key and Foreign key
35. $ What is Cross join, Union, Union All and other Joins in SQL
36. Why you prefer writing Typescript rather than Javascript
37. $ Observable concept
38. $ How React state works
39. $ What is subscribe
40. $ When you should unsubscribe
41. What is Redux and how it works.
42. $ Authentication in .Net (JWT) and practice
43. How to validate screen fields
44. What is Big Data Types in .Net?
45. Difference between Value Type and Reference Type in behaviour?
46. $ Despite the String is a reference type it passed by value,, What is the secret?
47. How dictionary data type works? How many operations does .net talks to search for an item inside dictionary? and what is the complexity?
48. If you have a List<int>, Where should it be stored? and If you have 1 million item in this list where do it stored and are we have a limitations in memory?
49. Which kind of abstractions you use in C#?
50. What is the concrete class?
51. Relation between abstraction and D Dependency Invertion Principle in SOLID? is the D work against the abstraction?
52. $ If you have a memory leackage in IIS, How would you troubleshhot this problem?
53. $ What is the SQL Dead Lock?
54. What is the execution plan for sql query?
55. $ Where Value, Reference Type stored in memory?
56. How to implement MIGS payment?