OOP, Java & C# Interview Questions and Answers

---

1. **List of 4 Pillars of OOP**

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

---

2. **What is Encapsulation?**

   Encapsulation means binding data and methods together in a single unit (class). It hides the internal data and exposes only necessary information.

Example:

```
class Student {
  private String name;
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
}
```

---

3. **What is Abstraction? Why do we use it?**

   Abstraction hides complex internal details and shows only essential features. It reduces complexity and improves code maintainability.

Example:

```
abstract class Vehicle {
  abstract void drive();
}
```

---

4. **Difference Between Encapsulation and Abstraction**

Encapsulation	**Abstraction**
Hides data by wrapping it into a single unit (class).	Hides complex implementation details and shows only essential features.
Achieved using access modifiers (private, public).	Achieved using abstract classes or interfaces.
Focuses on how to protect data.	Focuses on what to show or hide from the user.
Example: Private variables in a class	Example: Abstract methods or interfaces that define

with getters/setters.	behavior without implementation.

---

5. **What is Inheritance?**
   Inheritance allows a class to inherit properties and methods from another class.

Example:

```
class Animal { void eat() {} }
class Dog extends Animal { void bark() {} }
```

---

6. **Types of Inheritance**

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance (via interfaces in Java)
- Hybrid Inheritance

---

7. **What is Polymorphism?**
   Polymorphism means same method name behaves differently based on the context.

Example: Method Overloading and Overriding.

---

8. **Types of Polymorphism**

- Compile-time Polymorphism (Method Overloading)
- Run-time Polymorphism (Method Overriding)

---

9. **What is Interface? What’s the use of it?**
   An Interface is a contract that defines methods without implementation. It allows multiple classes to implement the same set of methods.

Example:

```
interface Animal { void sound(); }
```

---

10. **Difference Between Interface and Abstraction**

Interface	**Abstraction**

Defines a contract with only abstract methods (no implementation).	Can have both abstract and non-abstract methods.
Supports multiple inheritance.	Supports single inheritance only.
Cannot have method implementations (Java 7 and earlier).	Can have method implementations (concrete methods).
All methods are public and abstract by default.	Can define access modifiers for methods.
Example: interface Drawable in Java.	Example: abstract class Shape in Java.

---

11. **What is Marker Interface?**
    A Marker Interface has no methods or fields; it provides metadata information to the compiler or JVM.

Example: `Serializable` in Java.

---

12. **What is final keyword and its use?**
    The `final` keyword makes variables, methods, or classes unchangeable or non-overridable.

Example:

```
final int age = 30;
```

---

13. **What is static keyword and its use?**
    `static` means the member belongs to the class rather than any instance.

Example:

```
static int count;
```

---

14. **Difference between static and final keyword**

Static Keyword	**Final Keyword**
Belongs to the class rather than any object.	Value or method cannot be changed after assignment.
Used to create methods/variables common to all instances.	Used to declare constants or prevent method overriding.

Memory is allocated only once at the class level.	Once assigned, the variable value remains constant.
Example: static int count;	Example: final int MAX = 100;

---

15. **What is Lambda Function in Java?**
    - ➔ A lambda function in Java is a short way to write a method that can be passed as a parameter to another method.
    - ➔ It is mainly used to write simple, inline functions for operations like filtering, sorting, or iterating collections.

Example:

```java
Runnable r = () -> System.out.println("Hello");
```

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

numbers.forEach(n -> System.out.println(n));

---

16. **Difference Between throw and throws**

- `throw`: Manually throw an exception.
- `throws`: Declare that a method might throw an exception.

Example:

```java
throw new Exception("Error");
void method() throws Exception {}
```

Difference:

Aspect	**throw**	**throws**
Definition	Used to explicitly throw an exception.	Declares exceptions that a method might throw.
Usage **Location**	Used inside a method or block.	Used in method signature (after method name).
Purpose	To trigger an exception deliberately.	To declare possible exceptions for the caller.
Syntax	throw new ExceptionType();	returnType methodName() throws ExceptionType

Aspect	throw	throws
Example	throw new ArithmeticException("Error");	public void myMethod() throws IOException
Number of Exceptions	Can throw only one exception at a time.	Can declare multiple exceptions (comma-separated).
Type Required	Requires an instance of the Exception class.	Requires exception class names only.
Exception Type	Works with checked and unchecked exceptions.	Used mainly for checked exceptions.
Handling	The thrown exception should be handled using try-catch or propagated.	Declares that the method may throw an exception to be handled by the caller.

---

17. **What is Association and Generalization in Java?**

- **Association**: Relationship between two classes (Has-A).
- **Generalization**: Parent-Child relationship (IS-A).

Example: Association: Teacher - Student.
Generalization: Animal - Dog.

---

18. **What is Exception in Java? What's the use of it?**
    Exception is an error condition that can be handled to prevent program crash.

Example:

```
try { int a = 10/0; }
catch(Exception e) { System.out.println(e); }
```

---

19. **Types of Exceptions**

- Checked Exception (e.g., IOException)
- Unchecked Exception (e.g., ArithmeticException)

Type of Exception	Description	Examples

Type of Exception	Description	Examples
1. Checked Exceptions	Exceptions that are checked at compile-time. The programmer must handle them using try-catch or throws.	IOException, SQLException, FileNotFoundException
2. Unchecked Exceptions	Exceptions that are checked at runtime. These are due to programming errors and not mandatory to handle.	NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException
3. Errors	Serious problems that cannot be handled by the application. These are beyond the programmer's control.	OutOfMemoryError, StackOverflowError, VirtualMachineError

---

20. **System.out.println() Explanation**

- **System**: Predefined class.
- **out**: Static PrintStream object.
- **println()**: Method to print data.

---

21. **What is Singleton Class in Java and C#?**
    A Singleton class ensures only one instance is created.

Example:

```
class Singleton {
  private static Singleton instance = new Singleton();
  private Singleton() {}
  public static Singleton getInstance() { return instance; }
}
```

---

22. **What is Type Casting in C# and Java?**
    Converting one data type to another.

Example:

```
int a = (int) 10.5;
```

---

23. **What is Singleton, Scope, Transient in Java and C#?**

- **Singleton**: One instance for the whole application.
- **Scope (Scoped)**: Instance per request.
- **Transient**: New instance every time.

Example: Used in Dependency Injection in Spring (Java) or .NET Core.

Term	Definition	Example (Java/C#)
Singleton	A design pattern where only one instance of the class is created and shared.	@Singleton in Java, AddSingleton in C#
Scoped	A new instance is created per request or client session. Common in web apps.	AddScoped in C#
Transient	A new instance is created every time it is requested.	AddTransient in C#

Example in C#:

```
services.AddSingleton<IMyService, MyService>();

services.AddScoped<IMyService, MyService>();

services.AddTransient<IMyService, MyService>();
```

Example in Java (Spring Boot):

```
@Component

@Scope("singleton")

public class MySingletonService {}


@Component

@Scope("prototype") // Similar to Transient

public class MyPrototypeService {}
```

HTML, CSS, JavaScript Interview Questions and Answers

1. **What is HTML?**
   HTML stands for HyperText Markup Language. It is used to create the structure of web pages.

---

2. **What are semantic tags?**
   Tags that describe the meaning of content, like <header>, <footer>, <article>.

---

3. **What is the difference between <div> and <span>?**
   <div> is a block-level element, <span> is inline.

---

4. **What is the use of <meta> tag?**
   Provide metadata like charset, viewport settings for SEO and responsiveness.

---

5. **What is the difference between id and class?**
   Id is unique, used once per page. Class can be reused on multiple elements.

---

6. **What is Lazy Loading?**
   Instead of loading all the assets at once, the non-critical assets can be loaded on a need basis.

7. **What is a Doctype? Why is it important?**

   Defines the HTML version; helps browser render page correctly in standards mode.

---

8. **What is CSS?**
   CSS stands for Cascading Style Sheets. It is used to style HTML content like colors, fonts, and layouts.

---

9. **Difference between HTML and CSS?**

- HTML structures the content.
- CSS styles the content.

HTML	CSS
Stands for HyperText Markup Language.	Stands for Cascading Style Sheets.
Used to create the structure of web pages.	Used to style and design the web pages.
Deals with the content and layout of the page.	Deals with the presentation and look of the content.
Cannot style elements like color, font, spacing.	Allows customization of color, fonts, spacing, etc.
Example: <h1>Hello World</h1>	Example: h1 { color: blue; }

---

10. **Types of CSS, Priority, and Work:**
    - Inline CSS (Highest priority)
    - Internal CSS
    - External CSS (Lowest priority)

---

11. **What is the Box Model in CSS?**
    A model that wraps every element in Content, Padding, Border, and Margin.

---

12. **Difference between position: relative, absolute, fixed, and sticky?**
    - **relative**: Slightly shift from where it was supposed to be.
    - **absolute**: Place anywhere inside a container.
    - **fixed**: Always visible (e.g., navbar on top when scrolling).
    - **sticky**: Stays at the top while scrolling, but only when it reaches that point.

Property	Explanation	Position Based On
relative	Moves the element relative to its normal position without removing it from the flow.	Its original position
absolute	Removes the element from normal flow and positions it relative to the nearest positioned ancestor (other than static).	Nearest relative/absolute/fixed ancestor or the viewport if none
fixed	Removes the element from flow and positions it relative to the viewport, so it stays in the same	Viewport (browser window)

Property	Explanation	Position Based On
	place even when scrolling.	
sticky	The element toggles between relative and fixed based on the scroll. It sticks to a position when a certain scroll point is reached.	

---

13. **What are pseudo-classes in CSS?**
    Used to define a special state of an element,
    e.g., :hover, :focus, :nth-child().

---

14. Difference between em, rem, %, and px in CSS?
    - **px**: Fixed unit.
    - **em**: Relative to parent's font size.
    - **rem**: Relative to root (html) font size.
    - **%**: Relative to parent element.

---

15. **What is JavaScript?**
    JavaScript is Scripting language used to make web pages interactive and dynamic.

---

16. **What is the use of JavaScript in WebPages?**
    JavaScript adds interactivity and dynamic behavior to web pages.

---

17. **What are the different data types in JavaScript?**
    - String, Number, Boolean, Undefined, Null, Symbol, BigInt, Object.

---

18. **What is Hoisting in JS?**
    - Hoisting means JavaScript moves variable and function declarations to the top of the code before it runs.
    - This means you can use variables and functions before declaring them, but with some rules.
    - Accesible before Declaration :
        - Var : Yes, but value is undefine
        - Let : ReferenceError

- Const : ReferenceError
- Function : Fully Accesible

---

19. **What is use of Spread Operator in JavaScript?**
    The spread operator `...` expands an array or object into individual elements.

Example:

```
let arr = [1,2,3];
let newArr = [...arr, 4,5];
```

---

20. **What is Destructuring in JavaScript?**
    It is a syntax that allows you to unpack values from arrays or properties from objects into separate variables easily.

    Example-1:
    const numbers = [1, 2, 3];

    const [a, b, c] = numbers;
    console.log(a); // 1
    console.log(b); // 2
    console.log(c); // 3

    Example-2:
    const person = {
      name: "Krish",
      age: 25
    };

    const { name, age } = person;
    console.log(name); // Krish
    console.log(age);

---

21. **What is Difference Between == and ===?**
    - ==: compares values, does type coercion.
    - ===: compares value **and type strictly**.

---

22. **What are Arrow Functions?**

It is a shorter way to write functions in JavaScript using “=>” symbol.
It is useful for writing simple, cleaner, and faster functions, especially for small tasks.
Example: (a, b) => a + b

---

23. **What is a Callback Function?**

A callback function is a function that is passed as an argument to another function and is called later inside that function.

Real-world Example with setTimeout:

```
setTimeout(() => {
  console.log("This runs after 2 seconds");
}, 2000);
```

The function inside setTimeout is a **callback** that runs after 2 seconds.

Example:

```
function greet(name, callback) {
  console.log("Hello " + name);
  callback();
}

function sayBye() {
  console.log("Goodbye!");
}

greet("Krish", sayBye);
```

---

24. **What is the Event Loop?**

The event loop in JS handles asynchronous operations by managing the call stack and task queue.

Example :

```
console.log('1');

setTimeout(() => {
  console.log('2');
}, 0);

console.log('3');
```

output : 1 3 2

How it Work:

- JavaScript runs single-threaded code first (synchronous).
- Asynchronous tasks like setTimeout go to a queue.
- After main code is done, the Event Loop picks tasks from the queue and runs them one by one.

---

25. **Difference between Synchronous and Asynchronous JavaScript?**

Synchronous	**Asynchronous**
Code runs one line after another, step by step.	Code can start a task and move to the next line without waiting for the task to finish.
Each task waits for the previous one to finish.	Tasks can run in the background and complete later.
Blocking — next line won't run until the current one is done.	Non-blocking — doesn't stop the program while waiting.
Example: Simple for loop, math calculations.	Example: setTimeout, API calls, Promises.

---

26. **What are Promises in JS?**

A Promise represents the future completion of an asynchronous operation. It can be resolved, rejected, or pending.

Promise States:

- **Pending:** Initial state, operation is still running.
- **Fulfilled:** Operation completed successfully.
- **Rejected:** Operation failed.

Example :

```
let promise = new Promise((resolve, reject) => {
  let success = true;

  if(success) {
    resolve("Task completed successfully!");
  } else {
    reject("Task failed!");
  }
});
```

```
promise
.then(result => console.log(result))   // Success message
.catch(error => console.log(error));
```

---

27. **What is the difference between null and undefined?**
    - **null:** explicitly set by programmer, means no value.
    - **undefined:** means variable declared but not assigned.

    Example :
    ```
    let data = null;
    let info;
    console.log(data); // null
    console.log(info); // undefine
    ```

---

28. **Can we use super in JavaScript?**
    Yes, `super` is used to call parent class methods in subclass.

---

29. **What is the use of "this" keyword in JavaScript?**
    `this` refers to the current object context.

---

30. **What is Strict in JavaScript?**
    `'use strict'` enforces stricter parsing and error handling in JavaScript.

    Example:
    ```
    "use strict";

    x = 10;  // ✕ Error: x is not defined
    ```

---

31. **What is Call in JavaScript?**
    `call()` is used to invoke a function with a specific `this` value.1

    Example :
    ```
    const person = {
      fullName: function(city, country) {
        return this.firstName + " " + this.lastName + " from " + city + ", " + country;
      }
    };
    ```

```
const person1 = {
  firstName: "Krish",
  lastName: "Gohel"
};

console.log(person.fullName.call(person1, "Mumbai", "India"));
// Output: Krish Gohel from Mumbai, India
```

---

32. **What is map and filter in JavaScript?**

- `map`: Transforms each element in an array.
- `filter`: Filters elements based on a condition.

---

33. **Difference between map and filter:**

- `map` returns transformed array.
- `filter` returns only matching elements.

Map Function	**Filter Function**
Transforms each element in an array.	Filters elements based on a condition.
Returns a new array of the same length.	Returns a new array with fewer or same elements.
Example: array.map(x => x * 2)	Example: array.filter(x => x > 5)
Used for modifying data.	Used for extracting specific data.

---

34. **Difference between let vs var vs const:**

- `var`: Function scoped.
- `let`: Block scoped.
- `const`: Block scoped and cannot be reassigned.

Feature	**var**	**let**	**const**
Scope	Function-scoped	Block-scoped	Block-scoped
Reassignment	Allowed	Allowed	Not allowed
Hoisting	Hoisted and initialized with undefined	Hoisted but not initialized	Hoisted but not initialized
Redeclaration	Allowed within the same scope	Not allowed in the same scope	Not allowed in the same scope

Example	var a = 1;	let a = 1;	const a = 1;

---

35. **Name 5 Tags in HTML:**

- `<html>`, `<head>`, `<title>`, `<body>`, `<footer>`

36. **Name 5 Tags in Body:**

- `<h1>`, `<p>`, `<div>`, `<a>`, `<img>`

---

37. **Calculate and Display Count of** ** Tags:**

```
document.querySelectorAll('p').length;\
```

---

38. **Selectors in HTML, JavaScript, jQuery:**

- **HTML:** id, class, tag selectors.
- **JavaScript:** getElementById, querySelector.
- **jQuery:** $('#id'), $('.class'), $('tag')

---

39. **Password Show/Hide Toggle:**

```
function togglePassword() {
  var input = document.getElementById('password');
  if (input.type === 'password') input.type = 'text';
  else input.type = 'password';
}
```

---

40. **What is DOM in JavaScript?**

DOM (Document Object Model) is a tree structure representing the HTML elements of a webpage for dynamic manipulation.

---

41. **Explain Debouncing and Throttling.**

- **Debouncing :** Debouncing ensures that a function is called only after a certain delay once the event stops firing.
- **Throttling :** Throttling ensures that a function is called at regular intervals (e.g., once every 500ms) even if the event is triggered continuously.

Example of Debouncing :

function debounce(func, delay) {

```
    let timer;
    return function() {
        clearTimeout(timer);
        timer = setTimeout(func, delay);
    };
}

const processChange = debounce(() => {
    console.log('Input processed!');
}, 500);
```

Example of Throttling :

```
function throttle(func, limit) {
let lastFunc;
let lastRan;
return function() {
    const context = this;
    const args = arguments;
    if (!lastRan) {
        func.apply(context, args);
        lastRan = Date.now();
    } else {
        clearTimeout(lastFunc);
        lastFunc = setTimeout(function() {
            if ((Date.now() - lastRan) >= limit) {
                func.apply(context, args);
                lastRan = Date.now();
            }
          }, limit - (Date.now() - lastRan));
        }
    };
}


const processScroll = throttle(() => {
    console.log('Scroll event!');
}, 1000);
```

Difference :

Debouncing	**Throttling**
Executes after the event stops for a delay time.	Executes at fixed intervals during continuous events.
Best for input fields, search boxes.	Best for scrolling, resizing.
Prevents too many executions.	Limits executions to once per interval.

---

42. **What is the difference between LocalStorage, SessionStorage, and Cookies?**
    - **LocalStorage:** persists until manually cleared.
    - **SessionStorage:** cleared when tab closes.
    - **Cookies:** sent to server with requests, has expiry.

Feature	**LocalStorage**	**SessionStorage**	**Cookies**
Size	**~5MB**	**~5MB**	**~4KB**
Lifespan	Permanent until manually cleared	Cleared when the browser/tab is closed	You can set expiry time or session-based

Feature	LocalStorage	SessionStorage	Cookies
Scope	Accessible by all tabs & windows of the same origin	Accessible only in the tab/window where it's stored	Sent with every HTTP request
Stored On	Client-side only	Client-side only	Client-side + sent to server
Use Case	Save data like user preferences, theme settings	Temporary data like form inputs, one-time session data	Authentication, tracking user sessions
Access from Server	❌ Not sent to server	❌ Not sent to server	✅ Sent automatically with requests

---

SQL Interview Questions and Answers

1. **What is Join in SQL & How many types of Join?**

   Join is used to combine data from two or more tables based on related columns. Types of Joins:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- CROSS JOIN

---

2. **What is GROUP BY in SQL and its use?**

   GROUP BY groups rows with the same values in specified columns and is used with aggregate functions like COUNT, SUM, AVG.

   Example :

   SELECT Customer, SUM(Amount) AS TotalAmount

   FROM Sales

   GROUP BY Customer;

---

3. **When to use HAVING and WHERE in SQL?**

- WHERE: Filters records before grouping.
- HAVING: Filters records after grouping.

---

4. **Can we write aggregate function in WHERE condition?**

   No, aggregate functions can’t be used in WHERE; use HAVING instead.

---

5. **Difference Between HAVING and WHERE?**

WHERE Clause	**HAVING Clause**

Used to filter records before grouping.	Used to filter records after grouping (with GROUP BY).
Cannot use aggregate functions like COUNT, SUM.	Can use aggregate functions like COUNT, SUM.
Applies to individual rows in the table.	Applies to grouped records.
Example: WHERE salary > 50000	Example: HAVING COUNT(*) > 1

---

6. **What is the use of WITH in SQL?**

   WITH defines a Common Table Expression (CTE) for temporary result sets used within a query.

   It makes complex queries easier to read, write, and maintain.

   Example:

```
WITH CustomerTotal AS (
  SELECT
    CustomerName,
    SUM(Amount) AS TotalAmount
  FROM
    Orders
  GROUP BY
    CustomerName
)
SELECT * FROM CustomerTotal
WHERE TotalAmount > 500;
```

   Why Use WITH (CTE) :

- To simplify complex queries.
- To reuse the CTE in multiple parts of the query.
- Improves readability and modularizes queries.

7. **What is the use of TEMP in SQL?**

A Temporary Table in SQL is a table that is created and exists temporarily during a session or procedure.

It is useful for storing intermediate results temporarily.

Example :

-
  ```
  CREATE TABLE #TempEmployee (
      ID INT,
      Name VARCHAR(50),
      Salary INT
      );
  ```

-
  ```
  INSERT INTO #TempEmployee (ID, Name, Salary)
  VALUES (1, 'John', 5000),
  (2, 'Alice', 6000),
  (3, 'Bob', 5500);
  ```

-
  ```
  SELECT * FROM #TempEmployee;
  ```

-
  ```
  DROP TABLE #TempEmployee;
  ```

Type of Temparary Tables:

Type	**Scope**
#TempTable	Local, only visible in current session
##GlobalTempTable	Global, visible to all sessions until closed

8. **Difference Between WITH and TEMP:**

Aspect	**TEMP Table**	**WITH (CTE)**

Aspect	TEMP Table	WITH (CTE)
Definition	A **temporary physical table** stored in **tempdb** and available for the session or connection.	A **temporary result set (virtual table)** defined using the WITH keyword, used within a single query.
Scope	Exists for the **duration of the session or procedure**.	Exists **only for the duration of the query execution**.
Can be Indexed?	✅ Yes, you can create indexes on temp tables.	✖ No indexing allowed directly.
Can Store Data Permanently?	No, it's temporary but lasts longer than a CTE within the session.	No, exists only while the query is running.
Multiple Usage	✅ Can be queried **multiple times** in the session.	Usually used **once within the same query**.
Usage in Stored Procedure	Commonly used inside **stored procedures for complex logic**.	Less common inside procedures, mainly used to **simplify queries**.
Performance	May perform better with **large data sets and multiple queries**.	Suitable for **simpler, readable, and single-use queries**.

---

9. **What is Window Function in SQL?**

   Functions that perform calculations across a set of table rows related to the current row.

   Example:

```
SELECT
    Employee,
    Region,
    SalesAmount,
    RANK() OVER(PARTITION BY Region ORDER BY SalesAmount DESC) AS SalesRank
FROM
    Sales;
```

Output:

Employee	**Region**	**SalesAmount**	**SalesRank**
Carol	East	8000	1
Alice	East	7000	2
John	East	5000	3
David	West	6000	1
Eve	West	5500	2
Bob	West	4000	3

---

10. **List of Window Functions:**

- ROW_NUMBER()
- RANK()
- DENSE_RANK()
- NTILE()
- LAG(), LEAD()

11. **What is Stored Procedure? Usage and Example:**

- A Stored Procedure is a predefined set of SQL statements saved in the database that you can reuse anytime by calling it.
- It helps to automate tasks, improve performance, and keep business logic inside the database.

Key Points :

- Reduces code repetition.
- Increases security by controlling data access.
- Improves performance by reducing network traffic.
- Easy to maintain and update logic in one place.

CREATE PROCEDURE GetEmployees **AS BEGIN SELECT** * **FROM** Employees; **END**;

---

12. **Can we use Function instead of SP for insert, update, delete?**

No, Functions generally can't perform DML operations like insert, update, delete.

---

13. **Types of Functions in SQL:**

Function Type	**Description**	**Example Functions**
1. System Defined Functions	Predefined functions provided by SQL Server, Oracle, etc.	SUM(), AVG(), GETDATE(), LEN(), ROUND()
2. User Defined Functions (UDFs)	Functions created by users to perform custom operations.	Scalar, Inline Table-Valued, Multi-Statement Table-Valued

---

14. **What is User Defined Function? Example:**

Function created by user.

CREATE FUNCTION AddNumbers(@a INT, @b INT) RETURNS INT **AS BEGIN RETURN** @a + @b **END**

---

15. **What is SQL Injection? How to Protect?**

    SQL Injection is an attack to manipulate queries. Protect using:

- Parameterized queries
- ORM frameworks

---

16. **What is Index in SQL? Types of Indexing:**

    Index improves data retrieval speed. Types:

- Clustered Index
- Non-Clustered Index
- Unique Index

---

17. **What is Indexing and Sharding in SQL?**

- Indexing: Improves query speed.
- Sharding: Splitting data across multiple databases.

---

18. **Can we insert 0 or 1 into table via Function?**

    No, Functions can’t directly modify data in SQL Server.

---

19. **What is SubQuery in SQL? Example:**

    Query within another query.

SELECT * **FROM** Employees **WHERE** DeptID = (**SELECT ID FROM** Departments **WHERE** Name='HR')

---

20. **What is Key in SQL? Types:**

- Primary Key

- Foreign Key
- Unique Key
- Composite Key
- Candidate Key

---

21. **What is View in SQL?**

A virtual table based on the result-set of an SQL query.

Example :

```
- CREATE VIEW EmployeeView AS
  SELECT EmpName, Salary
  FROM Employees
  WHERE Salary > 5000;
- SELECT * FROM EmployeeView;
```

---

22. **What is Self join in SQL and it's Example.**
A Self Join in SQL is when a table is joined with itself.
It is used to compare rows within the same table using different alias of Table.

Example :

```
SELECT
   E.Name AS Employee,
   M.Name AS Manager
FROM
   Employee E
LEFT JOIN
   Employee M
ON
   E.ManagerID = M.EmployeeID;
```

---

23. **Can we use alternet key insted of foreign key for join in SQL?**
Yes, you can use an Alternate Key (Unique Key) instead of a Foreign Key when performing a JOIN in SQL, as long as the column you are joining on contains unique values in one of the tables.

Example :

Department Table:
DeptCode (Alternet Key) string
DeptName string

Employee Table:

EmpID	int
EmpName	string
DeptCode	string

```
SELECT e.EmpName, d.DeptName
FROM Employee e
JOIN Department d ON e.DeptCode = d.DeptCode;
```

SQL Query Interview Questions and Answers

1. **Query for employees with salary more than 50000:**

```
SELECT * FROM Employee WHERE Salary > 50000;
```

---

2. **Table structure for many-to-many relationship with single row output:**

   We can use STRING_AGG in SQL Server or GROUP_CONCAT in MySQL to combine degrees from multiple colleges. Example query:

```
SELECT Student, STRING_AGG(CollegeName + ': ' + Degree, ', ') AS CollegesDegrees
FROM StudentCollege
GROUP BY Student;
```

---

3. **Department wise Employee count where salary > 50000:**

```
SELECT Department, COUNT(*) AS EmployeeCount
FROM Employee
WHERE Salary > 50000
GROUP BY Department;
```

---

4. **Same as above but departments with count > 10:**

```
SELECT Department, COUNT(*) AS EmployeeCount
FROM Employee
WHERE Salary > 50000
GROUP BY Department
HAVING COUNT(*) > 10;
```

---

5. **Find second highest salary from employee:**

```
SELECT MAX(Salary) AS SecondHighestSalary
FROM Employee
WHERE Salary < (SELECT MAX(Salary) FROM Employee);
```

---

6. **Delete user where PermissionName = 'DeleteUser':**

```
DELETE FROM [User]
WHERE ID IN (
  SELECT ur.UserID
```

```
   FROM User_Role ur
   JOIN Role_Permission rp ON ur.RoleID = rp.RoleID
   JOIN Permission p ON rp.PermissionID = p.ID
   WHERE p.Name = 'DeleteUser'
);
```

---

7. **ProductName_CategoryName as Product:**

```
SELECT p.Name + '_' + c.Name AS Product
FROM Product p
JOIN Category c ON p.CategoryID = c.ID;
```

---

8. **Retrieve User data where Name contains given word:**

```
DECLARE @Word NVARCHAR(100) = 'inputWord';
SELECT * FROM [User]
WHERE Name LIKE '%' + @Word + '%';
```

MERN Interview Questions and Answers

ReactJS

1. **What is the role of proxy in React package.json?**

   the proxy field in package.json is used to proxy API requests from React frontend to the backend server during development.

Why Proxy is Needed

- React app typically runs on localhost:3000, and the backend (Node/Express) runs on localhost:5000.
- Browsers block cross-origin API calls due to CORS policy.
- Setting a proxy helps bypass CORS issues during development without changing backend CORS settings.

---

2. **What is React-Router and Its Types?**

   React-Router is a library in React to handle routing and navigation. Types:

- BrowserRouter
- HashRouter
- MemoryRouter

---

3. **What is Routing in React?**

   Routing in React means navigating between different components or pages without reloading the browser.

---

4. **Explain the use of React Router.**

   React Router is a standard routing library for React that enables navigation between different components in a single-page application (SPA) without reloading the entire page. It manages the URL and synchronizes it with the UI, ensuring a smooth and fast navigation experience.

   - **BrowserRouter**: Wraps the app to enable routing.
   - **Routes**: Contains all route definitions.
   - **Route**: Maps the path (/, /about, /contact) to the corresponding component.

- **Link**: Enables navigation without reloading the page.

---

5. **What is StrictMode in React?**

   StrictMode is a wrapper in React that helps identify potential problems in the application during development.

---

6. **What is State in React?**

   State is an object that holds data that can change over time in a component.

---

7. **What is Hook and Types of Hooks?**

   Hooks are functions that let you use React features in functional components. Types:

- useState
- useEffect
- useContext
- useRef
- useMemo

---

8. **What is useState and useEffect?**

- **useState:** Initializes and updates state in functional components.
- **useEffect:** Performs side effects like data fetching, updating DOM, etc.

---

9. **What is Props in React?**

   Props are short for properties. They are used to pass data from parent to child components.

---

10. **What is Class Component and Function Component?**

- **Class Component:** Written using ES6 classes, supports state and lifecycle methods.
- **Function Component:** Simple functions, can use hooks to manage state and side effects.

Aspect	Functional Component	Class Component
Definition	JavaScript function that returns JSX.	ES6 class that extends React.Component.
Syntax	Simpler and shorter.	More verbose with constructor and this.
State Management	Uses useState and other **Hooks**.	Uses this.state and this.setState().
Lifecycle Methods	Managed using **Hooks** like useEffect.	Has built-in lifecycle methods (componentDidMount, etc.).
this Keyword	Not used.	Must use this to access props, state.
Performance	Slightly faster and lighter.	Heavier due to more complex structure.
Readability	More concise and easier to read.	Can become complex with larger logic.
Example	See below.	See below.

---

11. **What is Callback Function?**

    A function passed as an argument to another function and executed later.

---

12. **What is Prop Drilling?**

    Passing data from parent to deeply nested child components through multiple layers.

---

13. **What is Virtual DOM and Real DOM? Difference:**

- **Virtual DOM:** A lightweight copy of the real DOM, helps in efficient updates.
- **Real DOM:** The actual DOM rendered on the browser.

Virtual DOM	Real DOM
A lightweight copy of the real DOM.	The actual DOM provided by the browser.
Changes do not directly update the screen.	Changes update the UI directly.
Faster performance due to batch updates.	Slower because each change updates the UI immediately.
Used in libraries like React for efficient rendering.	Used natively by browsers.
Works in memory and then updates Real DOM.	Directly represents the UI structure.

---

14. **What is Context API?**

    The Context API is a feature in React that allows you to share data (like state, themes, or authentication info) globally across the component tree without passing props manually at every level.

    It helps to avoid prop drilling, where you pass props through many nested components.

---

15. **What is Prop Drilling in React?**

    Prop Drilling is a situation where data (props) is passed from a parent component to deeply nested child components, even if intermediate components don't need that data — just to pass it down further.

    This can lead to:

- Unnecessary code complexity.
- Difficult to manage and maintain.
- Intermediate components become tightly coupled.

---

16. **What is the difference between Context API and Redux?**

Aspect	Context API	Redux

Aspect	Context API	Redux
Definition	Built-in React feature for global state management and avoiding prop drilling.	A state management library for managing complex application states.
Installation	No extra installation needed (built into React).	Requires installing redux and react-redux.
Boilerplate	Minimal code and setup.	More boilerplate with actions, reducers, store.
Usage Complexity	Simple and best for small to medium apps.	More structured, suited for large, complex apps.
State Structure	No strict rules; state is flexible.	Follows strict structure: store, reducer, action.
Middleware Support	Limited, manual implementation if needed.	Supports middleware like redux-thunk, redux-saga for async operations.
Performance	Re-renders all components consuming the context when value changes.	Fine-grained control over re-renders, better for large apps.
DevTools	No built-in dev tools.	Comes with Redux DevTools for state debugging.
Learning Curve	Easier and quicker to learn.	Steeper learning curve due to concepts like reducers, middleware.

---

17. **What is Memoization?**

Memoization is an optimization technique that caches the result of expensive function calls so that if the same inputs occur again, the cached result is returned instead of re-executing the function.

- Avoid **unnecessary re-renders**.
- Improve **performance** by **caching components or computed values**.
- **React.memo** : It is a Higher-Order Component (HOC) that wraps a functional component.
  - It prevents re-rendering if the props have not changed.
- useMemo : useMemo is a React Hook that memoizes a computed value.
  - It recalculates the value only when its dependencies change, avoiding expensive recalculations.

---

18. **What is the difference between Controlled and Uncontrolled Components?**

Aspect	**Controlled Component**	**Uncontrolled Component**
Definition	Form data is controlled by React state.	Form data is handled by the DOM directly.
Data Source	Data stored in React's state.	Data stored in the DOM (via ref).
Control	React has full control over form elements.	React has no direct control, only accesses data when needed.
Input Value Handling	Via value prop and onChange handler.	Accessed via ref to read the value.
Validation	Easy to implement validation, conditional rendering.	Harder to implement real-time validation.

Aspect	**Controlled Component**	**Uncontrolled Component**
Performance	Slightly less performant for very large forms due to state updates.	More performant in some cases, but less reactive.

19. **Difference Between Fetch API and Axios?**

Aspect	**Fetch API**	**Axios**
Type	Built-in JavaScript API (no installation).	External library (npm package).
Syntax	More verbose; requires manual handling of JSON.	Simpler and cleaner syntax.
Data Handling	Needs explicit response.json() to parse JSON.	Automatically parses JSON responses.
Browser Support	Native in modern browsers; may need a polyfill for older browsers.	Works in all modern browsers and Node.js.
Request/Response Interception	Not supported natively.	Supports interceptors for requests/responses.
Timeouts	No built-in timeout support.	Built-in timeout support.
Error Handling	Only rejects on network errors, not on HTTP errors (e.g., 404).	Rejects on HTTP errors and network errors.

Aspect	Fetch API	Axios
Cancel Requests	Supported via AbortController.	Supports cancellation using Cancel Token API.
Progress Tracking	Manual implementation required.	Supports progress tracking for uploads/downloads.
Size	Lightweight (native feature).	Slightly larger due to being a library.

Node.js

20. **What is the Event Loop in Node.js?**

The Event Loop in Node.js is a mechanism that handles asynchronous operations.

Since Node.js is single-threaded, the event loop allows it to handle multiple concurrent operations efficiently without blocking the main thread.

How It Works:

- **Call Stack:** Executes synchronous code first.
- **Task Queue:** Asynchronous callbacks are queued.
- **Event Loop:** Continuously checks the call stack; if empty, it takes tasks from the queue and pushes them to the stack for execution.

---

21. **Explain Stream and Buffers In Nodejs.**

- **Buffer:**

  - A Buffer in Node.js is a temporary memory storage for binary data.
  - It is used when dealing with raw data streams (like reading files, TCP streams, etc.).
  - Node.js introduced the Buffer class because JavaScript doesn't handle binary data well by default.

- **Streams:**

  - A Stream is a way to handle data piece by piece (chunks) rather than all at once.

- Useful when working with large files, network requests, or real-time data, where loading everything into memory isn't efficient.

---

22. **What is the Cluster Module in Node.js**

It allows you to create multiple instances (workers) of your Node.js application, enabling it to utilize multiple CPU cores.

Since Node.js runs on a single thread, the cluster module helps in achieving better performance and scalability by running multiple instances of the app on different cores.

ExpressJS

23. **What is Express.js and why is it used?**

- Express.js is a minimal, fast, and flexible web application framework for Node.js.
- It is widely used for building web servers, APIs, and backend services.
- Express simplifies the process of handling HTTP requests and responses by providing easy-to-use methods and middleware support.

---

24. **What is CORS and how do you enable it in Express?**

- CORS (Cross-Origin Resource Sharing) is a security feature implemented by browsers that controls how resources are shared between different origins (domain, protocol, or port).
- By default, a browser blocks requests made from one origin to a different origin (cross-origin) unless the server explicitly allows it via CORS headers.

---

25. **What is The Concept of Middleware Chaining?**

- Middleware chaining in Express refers to the process where multiple middleware functions are executed sequentially for a single request, passing control from one middleware to the next using the next() function.

Each middleware can:

- Process the request (`req`).
- Modify the response (`res`).
- Decide whether to pass control to the next middleware**.**
- End the request-response cycle.

MongoDB

26. **What is MongoDB? How is it different from SQL databases?**

MongoDB is a NoSQL, document-oriented database that stores data in a flexible, JSON-like format (called BSON).

- It is designed for **scalability, flexibility, and performance**, especially with large datasets and distributed systems.

- Instead of tables and rows (like SQL databases), MongoDB uses:

- **Databases ➞ Collections ➞ Documents**

---

27. **What is Aggregation in MongoDB?**

It is a process of transforming and combining data from multiple documents in a collection to return computed results.

- It works like SQL's GROUP BY and aggregate functions (like SUM, COUNT).

Stage	**Description**
$match	Filters documents (like WHERE in SQL).
$group	Groups data by a field and can perform aggregation like **sum, avg, count**.
$project	Selects specific fields to return, can rename or compute new fields.
$sort	Sorts documents in ascending or descending order.
$limit	Limits the number of documents returned.
$skip	Skips a specified number of documents.

---

28. **What is Embedded and Referenced relationships in MongoDB?**

- **Embedded (Denormalize) :**

  Store related data within the same document.

  Example:

```
{
    _id: 1,
    name: 'John',
    address: {
    street: '123 Main St',
    city: 'New York',
     zip: '10001'
    }
}
```

- **Referenced (Normalize) :**

  Store related data in separate documents with references (like foreign keys).

  Example:

```
{
  _id: 1,
  name: 'John',
  addressId: ObjectId("abcd1234")
}
```

---

29. **Difference Between populate() and aggregate() in Mongoose?**

Feature	**populate()**	**aggregate()**
Purpose	To join documents across collections based on references (like a SQL JOIN).	To perform data aggregation, transformation, and computation on documents (like GROUP BY, SUM, etc. in SQL).
Used For	Resolving referenced documents defined with ref in Mongoose schemas.	Performing complex queries, filtering, grouping, projections, calculations.
Ease of Use	Easier to use for basic joins/relations between collections.	More powerful and flexible, but more complex syntax.
Output	Returns documents with populated referenced fields.	Returns custom shaped data based on aggregation stages.
Performance	Faster and simpler for	More efficient for complex data

Feature	populate()	aggregate()
	small data sets.	processing but can be slower if not optimized.

---

30. **What is Pagination in MongoDB queries?**

    Pagination is used to fetch data in chunks (pages) instead of loading everything at once — useful for performance and UX.

    Example :

```
const page = 2;     // Current page number
const limit = 5;    // Number of records per page
const skip = (page - 1) * limit;


db.collection('users')
  .find({})
  .skip(skip)
  .limit(limit)
  .toArray()
  .then(users => console.log(users))
  .catch(err => console.error(err));
```

MongoDB Queries :

1. **Find Specific Field in All Documents.**

```
db.students.find({}, { name: 1, _id: 0 })
```

---

2. **Query with $and / $or Condition**

```
db.employees.find({
  $and: [{ age: { $gt: 25 } }, { department: "HR" }]
})
```

---

3. **$in and $nin Query**

```
db.products.find({ category: { $in: ["Electronics", "Clothing"] } })
```

Ex.Finds products in either Electronics or Clothing category.

4. **Update a Field using $set**

```
db.users.updateOne(
  { name: "John" },
  { $set: { status: "Active" } }
)
```

---

5. **Delete Documents**

```
db.orders.deleteMany({ status: "Cancelled" })
```

---

6. **MongoDB Aggregation for Grouping**

```
db.sales.aggregate([
  { $group: { _id: "$product", totalSales: { $sum: "$amount" } } }
])
```

Ex. Groups sales by product and calculates the total sales.

---

7. **Using $lookup for Joins**

```
db.orders.aggregate([
  {
    $lookup: {
      from: "customers",
      localField: "customerId",
      foreignField: "_id",
      as: "customerDetails"
    }
  }
])
```

Ex. Performs a join between orders and customers.

---

8. **Sorting Documents**

   db.users.find().sort({ age: -1 }) // Sort by age descending

---

9. **Pagination with limit and skip**

   db.products.find().skip(10).limit(5)

   Ex. Skip first 10 records and return next 5.

---

10. **Find Documents with Array Field**

    db.courses.find({ tags: { $all: ["javascript", "mongodb"] } })

    Ex. Finds courses where tags array contains both "javascript" and "mongodb".

Check Below Types Query From Google

- Aggregation pipelines
- Map-Reduce
- Text Search
- Index Query examples

.NET Interview Questions and Answers

1. **What is MVC in ASP.Net Core MVC?**

   MVC stands for Model-View-Controller, a design pattern that separates data (Model), UI (View), and business logic (Controller).

---

2. **Explain MVC lifecycle or pipeline.**

   - **Browser Sends Request**
     - A user requests a URL.
   - **Routing**
     - The request goes to the Routing Middleware.
     - It matches the URL to a Controller & Action.
   - **Controller Initialization**
     - The matched Controller is created.
     - Then the specific Action Method is called.
   - **Model Binding**
     - Data from the request (like query params, form data) is bound to method parameters or models.
   - **Action Execution**
     - The Action logic executes and returns a Result like a View or JSON.
   - **Result Execution**
     - The framework executes the result, e.g., returns a View, JSON, or Redirect.
   - **View Rendering (if applicable)**
     - If returning a View, it renders the HTML using the Razor View Engine.
   - **Response Sent**
     - The final response is sent back to the browser.

---

3. **What is Action Filter in ASP.Net MVC?**

   Action Filters are attributes used to add logic before or after an action method is called.

4. **Explain Action Filter lifecycle:**

- OnActionExecuting
- OnActionExecuted
- OnResultExecuting
- OnResultExecuted

---

5. **What is Middleware in .NET and explain its pipeline?**

   Middleware is a component in ASP.Net Core that handles HTTP requests and responses. Pipeline Example:

- Authentication Middleware
- Authorization Middleware
- Custom Middleware
- Endpoint Execution

---

6. **What is Dependency Injection?**

   Injecting object dependencies via constructor instead of creating objects inside classes.

---

7. **What is Web API in .NET?**

   Web API in .NET is used to build HTTP-based services like REST APIs.

   Type of it:

   - Rest API
   - SOAP API
   - GraphQL
   - OData
   - gRPC

---

8. **How to Pass List of Model from Controller to View?**

   Controller:

return View(modelList);

View:

@model List<**MyModel**>

Manipulate in View using Razor syntax.

---

9. **What is Boxing and Unboxing in .NET?**

- Boxing: Converting value type to object.
- Unboxing: Converting object back to value type.

Example:

- Boxing:

  int num = 10;

  object obj = num;

- UnBoxing:

  object obj = 20;   // Boxing

  int num = (int)obj;

---

10. **What is Type Casting in C#?**

    Converting a variable from one type to another.

int x = (int)myDouble;

---

11. **Types of ActionResult and Non-ActionResult:**

- ActionResult: ViewResult, JsonResult, RedirectResult
- Non-ActionResult: Custom return types like string, int

---

12. **What is the use of appsettings.json?**

Stores configuration data like connection strings, keys, and settings.

---

13. **How to maintain sessions in MVC?**

    Using:

- Session Variables
- TempData
- Cookies

---

14. **What is Partial View in MVC?**

    A reusable view rendered within another view.

```
@Html.Partial("_PartialView")
```

---

15. **How is the routing carried out in MVC?**

    Routes are defined in Program.cs under UseEndpoints. Example:

```
endpoints.MapControllerRoute(
   name: "default",
   pattern: "{controller=Home}/{action=Index}/{id?}"
);
```

---

16. **Difference Between TempData, ViewData, and ViewBag :**

Feature	**ViewData**	**ViewBag**	**TempData**
Type	Dictionary (string, object)	Dynamic property	Dictionary (string, object)
Scope	Available only in current request (same view)	Available only in current request	Available for current and next request
Persistence	Lost on Redirect	Lost on Redirect	Persists after Redirect

Feature	ViewData	ViewBag	TempData
			(till next request)
Usage Syntax	ViewData["Key"]	ViewBag.Key	TempData["Key"]
Example	ViewData["Message"] = "Hello";	ViewBag.Message = "Hello";	TempData["Message"] = "Hello";

---

17. **Differentiate ActionResult and ViewResult:**

Aspect	ActionResult	ViewResult
Definition	A base class that can return different types of results from a controller action.	A specific type of ActionResult that returns a View (HTML page).
Purpose	To return any kind of response (View, JSON, Redirect, etc.).	To specifically return a View (cshtml).
Example	return ActionResult (like JsonResult, RedirectToAction, ViewResult).	return View("Index");
Flexibility	More flexible, supports multiple types.	Limited to returning only Views.

---

18. **How is Spring Boot similar/different to ASP.Net Core?**

- Both are frameworks for building web apps and APIs.
- Spring Boot is Java-based, ASP.Net Core is .NET-based.
- Both support DI, middleware, REST APIs, and microservices.

---

19. **What is Razor Pages?**

Razor Pages is a simplified web development model in ASP.NET Core that combines HTML and C# code in a single page for building web applications.

---

20. **What is Peek and Keep in .Net?**

When using TempData in ASP.NET MVC or Core, data is usually removed after it's read in a request.

To persist TempData beyond the next request, we use:

- TempData.Peek()
  - Reads the value without removing it from TempData**.**
  - Useful if you want to read the data but keep it for future requests.
  - var value = TempData.Peek("Message");

- TempData.Keep()
  - Keeps specific TempData keys even after the request, so they are available for the next one too.
  - TempData.Keep("Message");

General Web/API/Architecture Interview Questions and Answers

1. **What is API?**

   API (Application Programming Interface) is a way for two software applications to communicate with each other.

---

2. **What is Middleware and How it works?**

   - Middleware is a software layer in an application pipeline that handles requests and responses.
   - It can perform operations like logging, authentication, authorization, and data transformation before passing to the next middleware or endpoint.

---

3. **What is JWT and what is its use?**

   - JWT (JSON Web Token) is a secure way to send information between two parties as a token (like client and server).
   - Commonly used for authentication to securely pass user information.

---

4. **What is Authentication and Authorization?**

- **Authentication:** Verifying who the user is.
- **Authorization:** Determining what an authenticated user is allowed to do.

---

5. **Difference Between Authentication and Authorization:**

Authentication	**Authorization**
Confirms the identity of the user.	Determines what resources the user can access.
Performed before authorization.	Performed after successful authentication.
Example: Logging in with username/password.	Example: Checking if user can access admin panel.
Ensures the user is who they say they are.	Ensures the user has permissions to do certain

	actions.
Always required in login systems.	Not always required if access control isn't needed.

---

6. **What is Client Side and Server Side Architecture?**

- **Client Side:** Code that runs in the user's browser (HTML, CSS, JS).
- **Server Side:** Code that runs on the server (Node.js, ASP.NET, Java).

---

7. **What is Client-Server Architecture?**

   - It's a model where clients send requests to a server, and the server processes and responds back.
   - The client is usually the browser or app, and the server is backend services.

---

8. **Types of Protocol in Browser:**

- HTTP
- HTTPS
- FTP
- WebSocket
- SPDY

---

9. **Difference between HTTP & HTTPS:**

HTTP	**HTTPS**
Stands for HyperText Transfer Protocol.	Stands for HyperText Transfer Protocol Secure.
Data is transferred in plain text.	Data is encrypted using SSL/TLS.
Less secure, prone to attacks like MITM (Man-In-The-Middle).	More secure, prevents data interception.
Uses port 80 by default.	Uses port 443 by default.

No SSL certificate required.	Requires an SSL certificate.
Suitable for non-sensitive data transfer.	Recommended for sensitive data like payments, login details.

---

10. **What is Status Code? And List Types of Status Codes:**

    Status codes are server responses to browser requests.

Types:

- 1xx: Informational (e.g., 100 Continue)
- 2xx: Success (e.g., 200 OK)
- 3xx: Redirection (e.g., 301 Moved Permanently)
- 4xx: Client Errors (e.g., 404 Not Found)
- 5xx: Server Errors (e.g., 500 Internal Server Error)

---

11. **Can you find data from JWT Token?**

    Yes, JWT tokens have three parts. The Payload part can be decoded and read but is not encrypted.

---

12. **List the Part Of JWT token? What's its use?**

- **Header:** Contains the algorithm and token type.
- **Payload:** Contains the user data or claims.
- **Signature:** Verifies that the token hasn't been tampered with.

---

13. **What is Session and Cookie and its use?**

- **Session:** Data stored on the server per user session.
- **Cookie:** Small data stored on the client's browser.

---

14. **Difference between Cookie and Session:**

Cookie	**Session**
Stored on the client-side (browser).	Stored on the server-side.
Limited storage capacity (around 4KB).	Can store larger amounts of data.
Less secure as data is stored on the client.	More secure as data stays on the server.
Data persists even after the browser is closed (until expiry).	Data is usually cleared when the session ends or browser closes.
Can be modified by the client.	Cannot be modified directly by the client.
Used mainly for tracking and storing user preferences.	Used for storing user-specific information like login sessions.

---

15. **Which algorithm is used for Bcrypt Password?**

Bcrypt uses the **Blowfish** cipher algorithm to securely hash passwords.

---

16. **What is RESTful API?**

RESTful API is a web service API design that uses standard HTTP methods (GET, POST, PUT, DELETE) to perform CRUD operations on resources over the web.

---

17. **What is Status Code? And List 6 Type of Status Code**

- 200 OK: Successful request
- 201 Created: Resource created
- 400 Bad Request: Client-side error
- 401 Unauthorized: Authentication required
- 404 Not Found: Resource not found
- 500 Internal Server Error: Server-side error

---

18. **What is the CI/CD pipeline?**
A CI/CD pipeline automates the build, test, and deployment of applications, ensuring faster and reliable deliver.

Stage	**Description**
1. Source Code Management	Code stored in GitHub, GitLab, Bitbucket, etc.
2. Continuous Integration (CI)	Automatically build and test code on every push/merge.
3. Continuous Deployment (CD)	Automatically deploy to production/staging servers.

---

19. **If you want to forgot the password then how you can work with JWT to Forgot password?**

- In forgot password feature, JWT can be used to generate a secure reset token that is sent to the user's email.
- This token ensures the link is safe, temporary, and unique.

Step's to Implement:

- **User Requests Password Reset:**
  - User submits their email ID**.**
  - Server checks if the email exists.
- **Generate a JWT Token**
  - Create a JWT token with user info and expiry time (e.g., 15-30 minutes).
  - var token = GenerateJwtToken(userId, expiresInMinutes: 30);
- **Send Reset Link via Email**
  - Email the user a link with the token:
    - https://yourapp.com/reset-password?token=JWT_TOKEN
- **User Clicks the Link**
  - The frontend reads the token from URL and sends it to the server.
- **Server Verifies Token**
  - Validate the JWT token:
    - Check signature & expiry.
    - Extract user ID from the token.
- **Allow Password Reset**

    - If token is valid, show the reset password form.
    - User sets a new password.
- **Update Password**
    - Update the password in the database and invalidate token (optional).

---

20. **Difference Restfull vs Restless vs Rest API's?**

- **REST API:** General term for APIs using REST.
- **RESTful API:** Proper and fully compliant REST API**.**
- **RESTless API:** Incorrect or partially REST API**,** often misusing rules.

Aspect	**REST API**	**RESTful API**	**RESTless API**
Definition	Any API based on REST principles.	An API that fully follows REST standards.	An API that partially follows or violates REST standards.
HTTP Methods	Uses HTTP methods like GET, POST, PUT, DELETE.	Uses correct HTTP methods for correct actions.	Often uses only POST or misuses HTTP methods.
URL Design	Resource-oriented but not always proper.	Strictly resource-oriented URLs (e.g., /users/1).	May use non-resource URLs (e.g., /getUser?id=1).
Statelessness	May or may not be stateless.	Always stateless (no server-side session).	Often stateful or session-dependent.
Standard Compliance	Can be partial or complete.	Fully compliant with REST principles.	Non-compliant or improperly designed.
Example	GET /users or POST /users	GET /products/5 to get product with ID 5.	POST /getProduct?id=5 instead of using GET.

Git and GitHub Interview Questions and Answers

1. **What is Git?**

   Git is a distributed version control system that helps track changes in source code during software development. It allows multiple developers to work together, manage versions, and roll back to previous code states.

2. **What is Git and GitHub?**

- **Git:** A tool to manage code versions locally and track changes.
- **GitHub:** A cloud-based hosting service that stores Git repositories online, enabling collaboration, sharing, and version control with teams.