All Workbook Questions

Web

Functional patterns

What is a `callback function`?

-> Answer: A callback function is a function that is passed as an argument to another function and is executed at a later point in time or in response to an event.
-> Example:

```
function calculateSum(a, b, callback) {
  const sum = a + b;
  callback(sum);
}

function displayResult(result) {
  console.log("The result is:", result);
}

calculateSum(3, 5, displayResult);
```

What is `ECMA script` ? What is the difference between Javascript & ECMA script ?

-> Answer: ECMAScript (ES) is a standardized scripting language specification that JavaScript is based on. It defines the syntax, semantics, and core features of the language.
JavaScript is an implementation of ECMAScript. It is the most popular and widely used implementation of the ECMAScript standard. Other implementations of ECMAScript include TypeScript, JScript, and ActionScript.
In essence, ECMAScript is the language specification, while JavaScript is the most prominent implementation of that specification.

What is the difference between `let` & `var` ?

-> Answer: The main difference between let and var is that let has block scope, while var has function scope. This means that a variable declared with let is only accessible within the block of code where it is defined, while a variable declared with var is accessible throughout the entire function in which it is defined.

-> Example:

```
function exampleFunction() {
  var x = 1;
  let y = 2;

  if (true) {
    var x = 3; // This modifies the value of the outer 'x'
    let y = 4; // This creates a new 'y' variable with block scope

    console.log(x); // Output: 3
    console.log(y); // Output: 4
  }

  console.log(x); // Output: 3 (value modified in the block)
  console.log(y); // Output: 2 (value remains unchanged outside the block)
}
```

Write an example where using the `var` declaration instead of the `let` could create a hard to debug code.

-> Example:

```
function exampleFunction() {
  for (var i = 0; i < 5; i++) {
    setTimeout(function() {
      console.log(i); // Logging the value of 'i'
    }, 1000);
  }
}
```

-> Answer: With var, the variable has function scope and is hoisted to the top of the function. This means that there is only a single i variable shared across all iterations of the loop.
As a result, when the setTimeout callbacks execute after the loop has finished, they all reference the same i variable, which by that time has reached the value of 5. As a consequence, all the logged values will be 5, regardless of the iteration.

Give a practical example where you would use the `reduce` function in javascript.

-> Example:

```
const numbers = [5, 10, 15, 20];

const sum = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue;
}, 0);

console.log(sum); // Output: 50
```

-> Answer: The reduce function iterates over each element in the numbers array and accumulates the sum of all the values. The initial value for the accumulator is set to 0. At each iteration, the callback function adds the current value to the accumulator, and the updated accumulator is passed to the next iteration. Finally, the reduce function returns the accumulated sum.

Give a practical example where you would use the `map` function in javascript.

-> Example:

```
const products = [
  { id: 1, name: 'Apple', price: 0.5 },
  { id: 2, name: 'Banana', price: 0.3 },
  { id: 3, name: 'Orange', price: 0.6 },
];

const productNames = products.map((product) => product.name);

console.log(productNames); // Output: ["Apple", "Banana", "Orange"]
```

-> Answer: By using map, you can transform the original array of objects into a new array containing only the desired property (name in this case). This can be useful when you need to extract specific data from an array or perform some kind of transformation on each element without modifying the original array.

Give a practical example where you would use the `filter` function in javascript.

-> Example:

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const oddNumbers = numbers.filter(number => number % 2 !== 0);

console.log(oddNumbers); //Output: [1,3,5,7,9]
```

-> Answer: In the callback function number => number % 2 !== 0, we check if the remainder of dividing each number by 2 is not equal to 0. If the remainder is not 0, it means the number is odd, and the callback function returns true for that number. As a result, the filter function creates a new array (oddNumbers) containing only the odd numbers from the original array.

Web basics

What is a `web server`?

-> Answer: A web server is a computer program or software that serves and delivers web pages and other web content to clients, usually web browsers, upon request.
It acts as the intermediary between the client and the requested web content, handling the incoming requests and providing the appropriate responses.
Web servers use the HTTP (Hypertext Transfer Protocol) protocol to communicate with clients and transfer data over the internet.
They store and manage web files, such as HTML documents, images, videos, and other resources, making them accessible to users through their web browsers.

Explain the `client-server architecture`.

-> Answer: Client-server architecture is a model where tasks are divided between a client and a server.
The client, such as a computer or smartphone, requests services or data from the server.
The server, a powerful computer or software, processes these requests and provides the requested information.
Communication occurs over a network, like the internet, and protocols like HTTP are used. This architecture enables distributed computing, efficient resource utilization, and centralized management of services and data.

What is the difference between `synchronous` and `asynchronous` execution?

-> Answer:

`Synchronous`: Synchronous execution refers to a mode of operation where tasks are performed sequentially and in order.
In this approach, a task must wait for the completion of the previous task before it can begin.
The execution flow is blocked until the current task finishes, and then the next task starts.
Synchronous execution ensures that tasks are carried out in a predictable and orderly manner.

`Asynchronous`: Asynchronous execution allows tasks to run independently and concurrently.
In this mode, a task can start execution without waiting for the completion of the previous task.
The execution flow is not blocked, and tasks can run in parallel or overlap.
Asynchronous execution is commonly used in scenarios where tasks can take variable time to complete, or when tasks can run concurrently without dependencies.

What is `npm`? Why is it useful?

-> Answer: `npm` is a powerful package manager for JavaScript projects, offering efficient package and dependency management, version control, and access to a wide range of community-driven packages.
It greatly simplifies the development process and enhances productivity for JavaScript and Node.js developers.

What is the difference between the `depdendencies` & `devDependencies` in a `package.json` file ?

-> Answer: `dependencies` includes packages necessary for the application to run correctly in a production environment, while `devDependencies` consists of packages required for development and testing tasks.
Separating these dependencies allows for a more streamlined and efficient deployment process, as production environments do not need to carry the additional overhead of development-related dependencies.

What would be the impact of javascript `fetch` if it was not asyncronous ?

-> Answer: If fetch were synchronous, the entire execution of the code would be paused until the network request is complete.
This blocking behavior would lead to unresponsive user interfaces, making the application appear frozen or slow.
Without asynchronicity, each network request would block the execution of other code, including event handling and UI updates.

Why benefits would bring to a developer to use the `Postman` application ?

-> Answer: Postman empowers developers by simplifying API testing, enhancing collaboration, automating workflows, facilitating documentation, offering monitoring and debugging capabilities, and integrating with other development tools.
These benefits contribute to improved productivity, efficiency, and the overall quality of API development.

List the `parts of the URL`.

-> Example:

```
 URL: https://www.example.com/path/to/resource?
param1=value1&param2=value2#section3
```

-> Answer:

```
 Scheme/Protocol: "https://"
 Host: "www.example.com"
 Port: (default port for HTTPS)
 Path: "/path/to/resource"
 Query Parameters: "param1=value1" and "param2=value2"
 Fragment/Anchor: "section3"
```

What is `query parameter`?

-> Example: `https://example.com/search?q=apple&category=fruits&page=1`

-> Answer: A query parameter, also known as a query string parameter, is a component of a URL (Uniform Resource Locator) that is used to pass information to a web server.
It is typically appended to the end of a URL and consists of a key-value pair.
The query parameter is preceded by a question mark "?" and multiple parameters are separated by an ampersand "&".

In this example, the query parameters are:

q=apple: The parameter key is "q" and the value is "apple".
category=fruits: The parameter key is "category" and the value is "fruits".
page=1: The parameter key is "page" and the value is "1".

What kind of `HTTP status codes` do you know?

-> Answer:

```
Informational (1xx):
100 Continue
101 Switching Protocols

Success (2xx):
200 OK
201 Created
204 No Content

Redirection (3xx):
301 Moved Permanently
302 Found
304 Not Modified

Client Error (4xx):
400 Bad Request
401 Unauthorized
404 Not Found
403 Forbidden

Server Error (5xx):
500 Internal Server Error
502 Bad Gateway
503 Service Unavailable
```

How does an `HTTP Request` look like? What are the most relevant `HTTP header fields`?

-> Example:

```
 GET /example HTTP/1.1
 Host: www.example.com
 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/91.0.4472.124 Safari/537.36
 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q
=0.8
 Accept-Language: en-US,en;q=0.9
 Cookie: session_id=123456789
```

-> Answer:

```
Request Line:
HTTP method: Specifies the action to be performed on the requested resource, such
as GET, POST, PUT, DELETE, etc.
Request target: The URL or path of the requested resource.
HTTP version: The version of the HTTP protocol being used, such as HTTP/1.1.

Headers:
 Host: Specifies the target host or domain name.
 User-Agent: Identifies the client making the request (e.g., the browser or
application).
 Accept: Indicates the media types (MIME types) acceptable in the response.
 Content-Type: Specifies the media type of the request's body (if present).
 Authorization: Provides credentials or authentication information for the
request.
 Cookie: Contains cookies associated with the request.
 Cache-Control: Defines caching directives for the request/response.
 Referer: Specifies the URL of the referring page.
 Accept-Language: Specifies the preferred language for the response.

Message Body (Optional):
Some HTTP requests, like POST or PUT, may include a message body that contains
data being sent to the server.
```

How does an HTTP Response look like? What are the most relevant HTTP header fields?

-> Example:

```
HTTP/1.1 200 OK </br>
Content-Type: text/html; charset=UTF-8 </br>
Content-Length: 1234 </br>
Cache-Control: max-age=3600 </br>
Set-Cookie: session_id=987654321; Expires=Wed, 30 May 2024 12:00:00 GMT </br>
Server: Apache/2.4.7 (Ubuntu)
```

-> Answer:

```
 Status Line: The status line is the first line of an HTTP response and contains
the following elements:

 HTTP version: The version of the HTTP protocol being used, such as HTTP/1.1.

 Status code: A three-digit numeric code that indicates the status of the
response.
 Common status codes include 200 (OK), 404 (Not Found), 500 (Internal Server
Error), etc.

 Reason phrase: A brief textual description associated with the status code.
 Example: HTTP/1.1 200 OK

Headers: HTTP headers provide additional information about the response and can
include various    fields.
Some of the most relevant HTTP header fields in a response are:

Content-Type: Specifies the media type of the response's body.
Content-Length: Indicates the length of the response's body in bytes.
Cache-Control: Defines caching directives for the response.
Set-Cookie: Sets a cookie in the client's browser.
Location: Specifies the URL to redirect to (in case of a redirect).
Server: Identifies the server software or platform.
Similar to HTTP requests, there are many other header fields available for
different purposes.

Message Body (Optional): The message body in an HTTP response contains the actual
content being sent by the server.
```

Why should you ignore the `node_modules` folder in `.gitignore` ?

-> Answer: Ignoring the node_modules folder in the .gitignore file helps keep the version control repository lean, improves performance, simplifies dependency management, ensures portability, and maintains a cleaner project structure.

Rest API: Serve and Fetch

Why is it recommend for a developer to use the http methods `get`, `put`, `delete` ?

-> Answer: Developers are recommended to use the HTTP methods GET, PUT, and DELETE for specific reasons related to the principles and conventions of the HTTP protocol and RESTful API design.

How does a `POST` request look like when it is made from a web browser (on the front end written) ?

-> Example:

```
POST /endpoint HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 34

{"key1": "value1", "key2": "value2"}

-> Answer: Request Line: It specifies the HTTP method (POST), the target endpoint
(/endpoint), and the version of HTTP being used (HTTP/1.1).

Headers: The headers provide additional information about the request. In this
example, we have a few common headers:

Host: It specifies the hostname of the server.

Content-Type: It indicates the type of data being sent in the request body.
   In this case, the content type is set to application/json, indicating that the
request body contains JSON data.

Content-Length: It specifies the length of the request body in bytes.

Request Body: It contains the data being sent with the request. In this example,
the request body is a JSON payload: {"key1": "value1", "key2": "value2"}.
```

What is an `API`?

-> Answer: API stands for **Application Programming Interface**.
API acts as a bridge between different software applications, enabling them to exchange information and perform actions on each other's behalf.
It provides a standardized way for developers to access and use the functionalities of a software system or service without needing to understand the underlying implementation details.

What is `REST API`?

-> Answer: REST API stands for **Representational State Transfer Application Programming Interface**.
In a REST API, resources are identified by unique URLs (Uniform Resource Locators), and the operations on these resources are performed using standard HTTP methods like GET, POST, PUT, PATCH, and DELETE.
REST APIs typically use the JSON (JavaScript Object Notation) format for data exchange, although other formats like XML can also be used.
They provide a standardized and efficient way for different systems to interact and exchange data.

What is `JSON` and how do we use it?

-> Answer: JSON stands for **JavaScipt Object notation**.
It is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate.
It is widely used for transmitting data between a server and a web application, as well as for storing and exchanging data between different systems.

What is `API versioning` ?

-> Answer: API versioning is the practice of assigning a unique identifier or number to different versions of an application programming interface (API).
It allows developers to make changes and updates to an API without breaking compatibility for existing users.
By versioning the API, developers can introduce new features, modify existing functionality, or fix issues while ensuring that existing integrations continue to work as expected.

Give 3 examples of `HTTP response status codes` ? Explain what each number means.

-> Answer:

```
200 OK:
The status code 200 OK indicates a successful HTTP request.
It means that the server has successfully processed the request, and the
requested resource or operation has been completed.
This is the most common status code for a successful response.
404 Not Found:
The status code 404 Not Found indicates that the server could not find the
requested resource.
It means that the URL or endpoint provided in the request does not correspond to
a valid resource on the server.
This status code is often used when a page or resource is removed or does not
exist.
500 Internal Server Error:
The status code 500 Internal Server Error is a generic server error response.
It indicates that there was an unexpected condition or error on the server that
prevented it from fulfilling the request.
This status code is typically used when the server encounters an error that it
cannot handle or when there is a bug in the server-side code.
```

Advanced JavaScript

How does the `ternary operator` looks like in javascript?

-> Example: `condition` ? `expression1` : `expression2`

-> Answer: `condition`: This is the expression that is evaluated. If the `condition` is true, `expression1` is executed; otherwise, `expression2` is executed.

`expression1`: This is the value or expression that is returned if the `condition` is true.

`expression2`: This is the value or expression that is returned if the `condition` is false.

How to import a function from another module in JavaScript?

-> Answer:

`ES6`: import { functionName } from './moduleFile.js';

`CommonJS(older method)`: const functionName = require('./moduleFile').functionName;

What is a shallow copy on an object?

-> Example:

```
const originalObj = {
  name: "John",
  age: 30,
  hobbies: ["reading", "swimming"]
};

const shallowCopy = Object.assign({}, originalObj);

shallowCopy.name = "Jane";
shallowCopy.hobbies.push("painting");

console.log(originalObj); //Outputs: { name: 'John', age: 30, hobbies: [
'reading', 'swimming', 'painting' ] }
console.log(shallowCopy); //Outputs: { name: 'Jane', age: 30, hobbies: [
'reading', 'swimming', 'painting' ] }
```

-> Answer: A shallow copy of an object is a new object that is created with the same properties as the original object.
However, the values of the properties are not recursively copied.
Instead, the properties of the new object are references to the same objects in memory as the original object.

What is a `callback function`? Tell some examples of its usage.

-> Answer: A `callback function` is a function that is passed as an argument to another function and is invoked or called within that function.

-> Example:

```
document.getElementById('myButton').addEventListener('click', function() {
  console.log('Button clicked!');
});
```

What is `object destructuring` in javascript?

-> Answer: Object destructuring is a feature in JavaScript that allows you to extract properties from an object and assign them to individual variables.

-> Example:

```
const person = {
  name: 'John',
  age: 30
};

const { name, age, city = 'Unknown' } = person;

console.log(name);  // Output: John
console.log(age);   // Output: 30
console.log(city);  // Output: Unknown
```

What is `array destructuring` in javascript?

-> Answer: Array destructuring is a feature in JavaScript that allows you to extract elements from an array and assign them to individual variables.

-> Example:

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

// Destructuring with rest syntax
const [a, b, ...rest] = numbers;

console.log(a);     // Output: 1
console.log(b);     // Output: 2
console.log(rest);  // Output: [3, 4, 5]
```

What is the spread operator in `js` ?

-> Answer: The spread operator is a syntax introduced in JavaScript that allows the expansion of an iterable (like an array or a string) into individual elements.
It is denoted by three consecutive dots (...).

What are the differences between the `arrow` function and the regular `function`?

-> Answer: The `arrow` function has a concise syntax with an `arrow` (=>) between the function parameters (if any) and the function body.
The `regular` function declaration has the function keyword followed by the function name, parameters (if any), and the function body enclosed in curly braces.
`regular` functions are hoisted and they can be called before initialization, but `arrow` functions cannot be.
If an `arrow` function consists of a single expression, it will automatically return the result of that expression without the need for explicit return keyword or curly brackets {}.

What is the `import` keyword used for?

-> Answer: To import functionality from other JavaScript modules.
It allows you to access and use code defined in separate modules within your current module.

-> Example: import { functionName } from './module.js';

What is the `required` used for?

-> Answer: The require keyword is used in Node.js to import functionality from other modules.
It is the counterpart to the import keyword used in ECMAScript modules.

-> Example: const module = require('./module.js');

What are `template literals`?

-> Answer: Template literals are enclosed by backticks ( ) instead of single or double quotes used for regular strings.
Within the template literal, you can embed placeholders ${expression} that are replaced with the evaluated value of the expression.

-> Example:

```
const name = 'John';
const age = 30;

const message = `My name is ${name} and I'm ${age} years old.`;

console.log(message);
// Output: My name is John and I'm 30 years old.
```

React basics

What benefits does it bring for a developer to use `components` (opposed of writing all the code in a single file) ?

-> Answer:
Using components in React promotes:

code reusability, modularity,

separation of concerns, abstraction,

collaboration, and scalability.

These benefits contribute to better code organization, productivity, and maintainability, making development more efficient and enjoyable for developers.

What is the difference between `Element` and `Component`?

-> Answer: An `element` represents a specific instance of a UI `component` and describes what should be rendered, while a `component` is a reusable and self-contained unit of code that encapsulates the logic and rendering of a part of the user interface.
`Elements` are the result of rendering `components`, and `components` are the building blocks that define the structure and behavior of the UI.

How do you pass values between components in `react`?

-> Answer:
`Props (Properties)`: Props are a way to pass data from a parent component to a child component.
The parent component can pass values or functions as props, and the child component can access and use those values within its own rendering logic or pass them further down to its own child components.
Props are read-only and cannot be modified by the child component. Here's an example:

`State`: State is used to manage and maintain internal data within a component.
State can be defined and updated within a component using the useState hook (for functional components) or the this.state and this.setState methods (for class components).
Components can use their own state or receive state values as props to render their UI or trigger actions based on changes.

What is `key` prop?

-> Answer: In React, the key prop is a special attribute used to uniquely identify elements in a collection of sibling elements.
When rendering a list or an array of elements, React requires each element to have a unique key prop assigned to it.

How does a child component pass data to it's parent component ?

-> Answer: In React, data can be passed from a child component to its parent component by using callback functions.
The parent component can define a callback function and pass it as a prop to the child component.
The child component can then invoke this callback function and pass data as an argument to communicate with the parent.

Write the code to create in JSX an HTML DIV element that has the innerText the contents of the variable `let name = 'Andrew'`

-> Answer:

```
let name = 'Andrew';

const element = <div>{name}</div>;
```

Write the code to create in JSX an unordered list from the array `let names = ["Mathew", "John", "Maverik"]`

-> Answer:

```
let names = ["Mathew", "John", "Maverik"];

const list = (
  <ul>
    {names.map((name, index) => (
      <li key={index}>{name}</li>
    ))}
  </ul>
);
```

Write the code to set the background color red of a div in JSX.

-> Answer:

```
const element = <div style={{background-color: 'red'}}>Hello, world!</div>;
```

React patterns

What is the difference between `Real DOM` and `Virtual DOM`?

-> Answer: `The Virtual DOM` serves as an optimization technique to enhance the performance of web applications by minimizing the direct manipulation of the `Real DOM` and reducing unnecessary updates.

When adding an item to an array, why is it necessary to pass a new array to the `useState` hook ?

-> Answer: In order to properly update the state and trigger a re-render of the component.
This is because React uses referential equality to determine if the state has changed and needs to be updated.

-> Example:

```
// Incorrect way (mutating the existing array)
const [items, setItems] = useState([]);

const addItem = (item) => {
  items.push(item); // Modifying the existing array directly
  setItems(items); // This would not trigger a re-render
};

// Correct way (creating a new array)
const [items, setItems] = useState([]);

const addItem = (item) => {
  const newItems = [...items, item]; // Creating a new array with the added item
  setItems(newItems); // This triggers a re-render
};
```

Describe what techniques or tools you use to debug a react app.

-> Answer: Console.log: The simplest and most widely used debugging technique is adding console.log statements at various points in your code to output values or track the flow of execution.
You can log state, props, function calls, or any other relevant information to the console.

What is the difference between a react `class` component & a `functional` component ?

-> Answer: `Class components` are created using JavaScript classes and have their own state, lifecycle methods, and instance (this).
They are used for managing state and using lifecycle methods.

`Functional components` are defined as JavaScript functions and don't have their own state or lifecycle methods by default.
However, with React Hooks, they can now have state and use lifecycle functionality.
They are simpler, more concise, and recommended for most use cases.

Name 3 lifecycle states in a react `functional` component.

-> Answer:
`useEffect`: This hook allows you to perform side effects in your functional component.
It is similar to the lifecycle methods componentDidMount, componentDidUpdate, and componentWillUnmount in class components.
With useEffect, you can subscribe to events, fetch data, update the DOM, or perform other actions after rendering.

`useState`: This hook enables you to add state to your functional component.
It replaces the this.state and this.setState functionality in class components.
With useState, you can declare and update state variables within the function body.

`useContext`: This hook provides access to the value of a React context within a functional component.
It allows you to consume context values without needing to use a Context.Consumer component or wrapping your component in a Context.Provider.

What is conditional rendering in `react` ? Give an example.

-> Answer: It allows you to dynamically control what is displayed in your UI based on the state of your application.

-> Example:

```
import React from 'react';

function ExampleComponent() {
  const isLoggedIn = false;

  return (
    <div>
      {isLoggedIn ? (
        <h1>Welcome, User!</h1>
      ) : (
        <h1>Please log in to continue.</h1>
      )}
    </div>
  );
}
```

Write the code that prints to the console `component destroyed` when the component it is part of is removed from the DOM tree.

-> Answer:

```
import React, { useEffect } from 'react';

function ExampleComponent() {
  useEffect(() => {
    return () => {
      console.log('component destroyed');
    };
  }, []);

  return (
    // Component's JSX content
    <div>
      {/* Content of the component */}
    </div>
  );
}
```

Why is there an infinite loop in this code

```
function App() {
  const [count, setCount] = useState(0); //initial value of this
  useEffect(() => {
    setCount((count) => count + 1); //increment this Hook
  }); //no dependency array.
  return (
    <div className="App">
      <p> value of count: {count} </p>
    </div>
  );
}
```

-> Answer: When a dependency array is not specified, the effect runs after every render of the component.
setCount((count) => count + 1). This update triggers a re-render of the component, which in turn executes the effect again, causing an infinite loop of re-renders and updates.

Why is there an infinite loop in this code

```
  async function App() {
  const [count, setCount] = useState("");
  setCount(count + 1);
  return (
    <div className="App">
      <p> value of count: {count} </p>
    </div>
  );
}
```

-> Answer: In React, state updates using useState should only be performed inside functions or callbacks that are triggered by user interactions, events, or lifecycle methods.
Directly updating the state outside these designated places can lead to unexpected behavior, including infinite loops.

Mongo & mongoose

What is a `database schema` ?

-> Answer: It defines the structure of documents within a collection.
 MongoDB's flexible schema allows for dynamic data models, meaning each document within a collection can have different fields and structures.

Why is the `id` unique in a database ?

-> Answer: The "id" field in a database is typically designed to be unique to ensure data integrity and facilitate efficient data retrieval and manipulation.

What are the advatanges & disadvatages of using `lean()` function in a `mongo query` ?

-> Answer: Using lean() can offer performance improvements and reduced memory usage, especially for read-heavy operations with large result sets.
However, it comes with limitations such as the loss of Mongoose-specific features, change tracking, schema validation, and population capabilities.

Write the code to store the object `{name: "Andrew", age: 10}` to a mongo database. You can ignore the part of connection parameters.

```
const mongoose = require('mongoose');
// Define the schema for the object
const exampleSchema = new mongoose.Schema({
  name: String,
  age: Number
});
// Create a model based on the schema
const ExampleModel = mongoose.model('Example', exampleSchema);
// Connect to MongoDB (Assuming you have already set up the connection)

// Create a new instance of the model with the data
const exampleObject = new ExampleModel({
  name: "Andrew",
  age: 10
});

// Save the object to the database
exampleObject.save()
  .then(savedObject => {
    console.log('Object saved to the database:', savedObject);
  })
  .catch(error => {
    console.error('Error saving object:', error);
  });
```

Write the code to `delete` from a mongo database all employees that are over 18 years. The scheme for an employee is `{name: string, age: int}`. You can ignore the part of connection parameters.

```
const mongoose = require('mongoose');
// Define the schema for the employee
const employeeSchema = new mongoose.Schema({
  name: String,
  age: Number
});
// Create a model based on the schema
const EmployeeModel = mongoose.model('Employee', employeeSchema);
// Connect to MongoDB (Assuming you have already set up the connection)
// Delete employees over 18 years
EmployeeModel.deleteMany({ age: { $gt: 18 } })
  .then(deletedEmployees => {
    console.log('Deleted employees:', deletedEmployees);
  })
  .catch(error => {
    console.error('Error deleting employees:', error);
  });
//The { age: { $gt: 18 } } query condition selects all employees with an age
greater than 18.
```

Write the code to `display in the console` from a mongo database the employees who are over 18 years. The scheme for an employee is `{name: string, age: int}`. You can ignore the part of connection parameters.

```
const mongoose = require('mongoose');
// Define the schema for the employee
const employeeSchema = new mongoose.Schema({
  name: String,
  age: Number
});
// Create a model based on the schema
const EmployeeModel = mongoose.model('Employee', employeeSchema);
// Connect to MongoDB (Assuming you have already set up the connection)
// Find employees over 18 years
EmployeeModel.find({ age: { $gt: 18 } })
  .then(employees => {
    employees.forEach(employee => {
      console.log('Employee:', employee);
    });
  })
  .catch(error => {
    console.error('Error retrieving employees:', error);
  });
```

Write the code to `update` from a mongo database the employees with name='John' and set the new age to 8. The scheme for an employee is `{name: string, age: int}`. You can ignore the part of connection parameters.

-> Answer:

```
const mongoose = require('mongoose');

// Define the schema for the employee
const employeeSchema = new mongoose.Schema({
  name: String,
  age: Number
});

// Create a model based on the schema
const EmployeeModel = mongoose.model('Employee', employeeSchema);

// Connect to MongoDB (Assuming you have already set up the connection)

// Update employees with name='John' to new age 8
EmployeeModel.updateMany({ name: 'John' }, { age: 8 })
  .then(updatedEmployees => {
    console.log('Updated employees:', updatedEmployees);
  })
  .catch(error => {
    console.error('Error updating employees:', error);
  });
```

Mern stack

What does `MERN` stand for in the context of web development ?

-> Answer:
`M: MongoDB` - A popular NoSQL database that stores data in a flexible, JSON-like format.

`E: Express.js` - A web application framework for Node.js that provides a set of tools and features to build server-side applications and APIs.

`R: React` - A JavaScript library for building user interfaces. React allows developers to create reusable UI components and efficiently update the user interface as the application state changes.

`N: Node.js` - A JavaScript runtime environment that allows developers to run JavaScript on the server-side. Node.js provides a platform for building scalable and high-performance server applications.

What is routing in the context of a `react` app ?

-> Answer: Routing refers to the process of managing navigation and rendering different components based on the current URL or route.
React Router is a popular library used for implementing routing functionality in React applications.

What is routing in the context of an `express` app ?

-> Answer: Routing refers to the process of defining routes and handling HTTP requests that are received by the server.
Express is a popular web application framework for Node.js that simplifies the creation of server-side applications and APIs.

What is `CORS` policy ?

-> Answer: CORS stands for Cross-Origin Resource Sharing.
It is a security mechanism implemented by web browsers that enforces restrictions on cross-origin requests made by web applications.
The CORS policy is a set of rules defined by the server that specifies which origins (domains) are allowed to access its resources.

What advantages does a developer have for using `bootstrap` or `material ui` ?

-> Answer: Bootstrap and Material-UI offer a rich set of UI components, responsive design features, and customization options.
The choice between them depends on the specific project requirements, design preferences, and familiarity with the respective frameworks.

OOP

.NET Framework

What is `.NET`?

.NET is an `open source development platform` (languages: C#, F#, VB; + libraries) for building different types of apps.
Its implementations are .NET Core (Windows, Linux, macOS), .NET Framework (Windows) and Xamarin (mobile) for different platforms.

Can you describe the difference between `.NET Framework` and `.NET Core`?

Both are used to develop C# (and F#, VB) applications, but `.NET Framework` has been around longer,
is maintained by Microsoft and can only be used to develop for Windows.
The `.NET Core` (meanwhile named only ".NET"),
also developed by Microsoft is a free and open source framework to develop applications for Windows, Linux and macOS.

What project types do you know in `Visual Studio/Rider`?

`Class Library, Console Application, Desktop Application, Unit Test Project, ASP.NET Core Web App.`

What is a `solution`?

A solution is a container to organize one or more related code projects. A solution contains a project for each build output (dll, exe, msi).

What is an `assembly`?

An assembly is a basic building block of a .NET application. It is a file that is automatically generated by
the compiler upon successful compilation of every .NET application. It can be a DLL (dynamic link library) or an executable file.

What is `LINQ` in .NET? How does it work?

LINQ stands for **Language-Integrated Query** which adds native data querying capabilities to .NET languages.
The operators defined by LINQ are exposed to the user through the **Standard Query Operator API.**
The query expressions are similar to SQL statements and can be used to extract and process data from arrays,
enumerable classes, XML documents, relational databases.

What are some commonly used `LINQ methods` that you know?

`Select, Where, SelectMany, Sum, Min, Max, Average, Aggregate, Take, Skip, OfType, Concat, OrderBy, GroupBy, Distinct, Contains, Count.`

Which .NET class would you use if you need to `generate random numbers`? Explain its usage briefly.

The `System.Random` class to generate pseudorandom numbers. An instance of the Random class has to be created, then its methods can be used to generate random numbers. Creating an instance of Random should not be done inside a method, its especially dangerous within a cycle, because of how the class works.

The class uses a seed number in order to generate sequences of random numbers. When two instances are generated with the same seed number, the "random" numbers will be the same.

```
Random random = new Random();
int myRandomNumber = random.Next(); // Generates signed integer from 0 to Int32 Maxvalue
int myRandomNumberUpToNine = random.Next(10); // Generates integer between 0 and 10 (exclusive)
double myRandomDouble = random.NextDouble(); // Generates floating point number between 0.0 and 1.0 (exclusive)
```

Which .NET classes are used to `read and write files in C#`?

`Classes of the System.IO Namespace, StreamReader and StreamWriter and methods of the File class.`

Which .NET type is used for `working with dates`? Describe some of the functions related to this type.

The `DateTime` class is used to work with dates. Methods include: Add methods (e.g. AddHours or AddDays),
`.Date` to access the date part, `.TimeOfDay` to access the time part, etc.

Which .NET class can be used for `measuring time`?

The `Stopwatch` class can be used to measure elapsed time. Its `Start` method has to be invoked to start it and its `Stop` method to stop it. One of the `Elapsed` methods is used to get the elapsed time.

What is `NuGet`?

NuGet is a package manager for .NET to create, share and consume useful .NET libraries. It was introduced in 2010 as NuPack.

What is the `IEnumerable` interface?

The `IEnumerable` interface allows classes that implement it to use enumeration (iteration) over a collection.

What does the term *`deferred execution`* mean? How does it relate to `IEnumerable` and `LINQ`?

In relation to LINQ it means that when assigning a variable using a LINQ query, the query will not be run until the value of the variable is actually required.

Its implication is that on each subsequent access of the variable, the evaluation runs again on the latest data. This is called **lazy evaluation**.

Describe some `collection types` you know.

1. Array: An array is a fixed-size collection of elements of the same type. It provides fast access to elements using an index. The size of an array is determined at the time of its creation and cannot be changed.
2. List: List is a dynamic data structure that can grow or shrink in size. It allows storing elements of different types and provides methods for adding, removing, and accessing elements. Elements in a list are accessed using their index.
3. HashSet: HashSet is an unordered collection of unique elements. It does not allow duplicate values and provides fast operations for adding, removing, and checking the presence of an element. The order of elements in a HashSet is not guaranteed.
4. Queue: Queue is a data structure that follows the First-In-First-Out (FIFO) principle. Elements are added to the back of the queue and removed from the front. It supports operations like enqueue (add to the back) and dequeue (remove from the front).
5. Stack: Stack is a data structure that follows the Last-In-First-Out (LIFO) principle. Elements are added to the top of the stack and removed from the top. It supports operations like push (add to the top) and pop (remove from the top).
6. Dictionary: Dictionary is a collection of key-value pairs. It provides fast lookup based on the key and allows storing unique keys. Keys are used to access and retrieve corresponding values efficiently.
7. SortedList: SortedList is similar to a Dictionary, but it maintains the elements in sorted order based on the key. It provides efficient key-based lookup and supports operations like adding, removing, and accessing elements.
8. SortedSet: SortedSet is a collection of unique elements that are sorted in ascending order. It provides fast operations for adding, removing, and checking the presence of an element. The order of elements in a SortedSet is guaranteed.
9. Tuple: Tuple is a lightweight data structure that can store a fixed number of elements of different types. It allows grouping multiple values together and provides a convenient way to return multiple values from a method.

Why does the `System.String` type implement the `IEnumerable` interface? What are the advantages of this?

The `System.String` type implements `IEnumerable` because in the background System.String is a `char[]`.
The advantages are that individual characters of a string can be accessed more easily and the string is iterable.

Language features

What `control statements` are available in C#?

It's possible to create labels and use the `goto` keyword to jump to specific labels.

Calling a subroutine (function, method) can also be counted as a control statement together with its `return` keyword.

Conditional statements are widely used, they include `if-then-else` statement, which can be written with the ternary operator also.

If there are a high number of cases, the `switch-case` statement allows to write compact code.

A loop is yet and other of control statement, including `for`, `while`, `foreach` loops.

It is possible to exit a loop early with `break` command or to skip parts of code in the loop with the `continue` command.

What is the difference between a `for` loop and a `foreach` loop?

The `foreach` loop has to be used on a collection that implements the `IEnumerable` interface. It then enumerates through all the elements
of the collection from start to finish (given that the `break` keyword is not used), while a `for` loop is a more general loop,
a collection is not needed, and the exact logic of enumeration must be provided to the `for` loop. E.g:

```
int[] numbers = {1, 2, 3, 4, 5};
foreach (int number in numbers) //every number is enumerated and printed
{
    Console.WriteLine(number);
}

for (int i = 0; i < numbers.Length; i += 2) //every number with odd index is
enumerated and printed
{
    Console.WriteLine(numbers[i]);
}
```

What is a `while` loop?

A `while` loop is a control flow statement that allows code to be executed repeatedly based on a (boolean) condition.

It can be thought of as a repeating `if` statement.

What does the `yield` keyword do?

The `yield` keyword returns elements one element at a time from an enumerator block (for, foreach, while) as an `IEnumerable` collection.

It uses requires less memory and less code than creating the IEnumerable before the enumerator and filling it in the enumerator, and returning after.

How do you manually break out of a loop?

Using the `break` (`yield break`), `return`, `goto`, `throw` keywords.

What does the `var` keyword mean?

The `var` keyword (stands for variable) and is used as a wildcard to not having to specify exact variable type of the variable.
With the `var` keyword, the type is implicitly specified. It should be used with a well named variable.

What is the *primary constructor syntax*?

The primary constructor syntax is a feature supposedly introduced in C# 9.0 that provides a concise way to declare immutable types, like records that was also introduced in C# 9.0.

The properties of the class are initialized in the constructor without needing to declare them explicitly.

Examples of not using and using primary constructor syntax:

```
//prior to C# 9.0
public class Person()
{
    public string Name { get; init; }
    public int Age { get; init; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
}

//with primary constructor syntax this becomes:
public record Person(string Name, int Age);
```

What is the meaning of the `params` keyword? Where would you use it?

In C#, params is a keyword which is used to specify a parameter that takes variable number of arguments.
It is useful when we don't know the number of arguments prior. Only one params keyword is allowed and no additional parameter is permitted after params keyword in a function declaration. For example:

```
public int TotalMarks(params int[] list)
{
    int total = 0;
    for (int i = 0; i < list.Length; i++)
        total += list[i];
    return total;
}
//following method calls work properly:
int total3 = Students.TotalMarks(8, 9, 8);
int[] marks = { 24, 22, 25, 21 };
int total4 = Students.TotalMarks(marks);
int total5 = Students.TotalMarks(92, 90, 95, 91, 98);
```

What are *lambda expressions*? How are they used in C# development?

Lambda expressions in C# are similar to arrow functions in JavaScript. They are basically anonymous functions with less code.

Expression lambda: `input => expression`, Statement Lambda: `input => { statements };`

Examples:

```
List<int> numbers = new List<int>() {36, 71, 12, 15, 29, 18, 27, 17, 9, 34};
var square = numbers.Select(x => x * x);
List<int> divBy3 = numbers.FindAll(x => (x % 3) == 0);
var ordered = numbers.OrderBy(x => x);
```

What is the difference between a jagged array and a multidimensional array?

In a `multidimensional array`, each element in each dimension has the same, fixed size as the other elements in that dimension.

In a `jagged array`, which is an array of arrays, each inner array can be of a different size. Examples:

```
public class ArrayHolder
{
    int[][] jaggedArray = { new int[] {1,2,3,4},
                            new int[] {5,6,7},
                            new int[] {8},
                            new int[] {9}
                          };

    int[,] multiDimArray = {{1,2,3,4},
                            {5,6,7,0},
                            {8,0,0,0},
                            {9,0,0,0}
                           };
}
```

Type system

What are `primitive types` in C#? Give some examples.

Primitive types are `int, object, short, char, float, double, char and bool`. These types are used to build other data types.

What is the difference between `value types` and `reference types`?

Variables of reference types store references to their data and multiple variables can have the same reference.
When the data on the reference object changes, since the variables only store the reference, the data in all
the variables changes. Reference types are stored on the heap. Reference types: class, interface, delegate,
record, dynamic, object, string.

With value types, each variable has its own copy of the data and the operations on one variables does not
affect the data on the other variable (except when using `ref`, `in`, `out` parameter variables). Value
types are stored on the stack. Value types: integral numeric types, floating-point numeric types, bool, char.

What is a `class` in C#?

A class is a `data structure` that may contain data members `(constants and fields)`, function members `(methods, properties, events, indexers, operators, instance constructors, finalizers and static constructors)` and nested types.

Class types support `inheritance`, a mechanism whereby a derived class can extend and specialize
a base class. A class is also often referred to as a blueprint for an object where the variables and methods
of an object are defined.

What is a `constructor`?

A constructor in C# is a member of a class. It is a method in the class which gets executed when a class
variable is created (a class is instantiated). Usually we put the initialization code in the constructor. The name of the constructor is always is the same name as the class. A C# constructor can be public or private.

Is it possible to have `multiple constructors in a class`?

Yes. These are known as `overloaded constructors`.

Each constructor can have a different parameter list, enabling the creation of objects in various ways or with different initializations.
When an object is instantiated, the **appropriate constructor is called based on the arguments provided or their absence**. This feature allows for **more flexibility** in **creating objects tailored to specific needs within the class**.

What are `properties` in C#?

A property is a class member through which reading (getting) and writing (setting) of a private field is possible.
Properties behave like public data members, but they implement special methods, called accessors (getter,
setter, init). Properties can be read-write, read-only or write-only.

Describe the different types of properties: `read-only, init-only, and computed properties`.

`Read-only properties` have to be set when an object is created, they may only be read after that, but not changed.

`Init-only properties` do not have to be set when the object is created, so the object may be created with the default
constructor and the property set later, but it might be only set once.

`Computed properties` are properties that may not only return a value of a member field, but can return any computed
combination of any of the private fields. For example FullName could return Firstname with LastName concatenated.

What is the difference between an `auto-property` and a `property with a backing field`?

With an auto-property a private backing field is auto-generated. If both have a getters and setters, and the backing field is private, then they are equivalent.

What is an `enum` in C#?

Enum is a special class that represents a group of constants. By default the first item in an enum has the
value of 0. Enums should be used when there are values that aren't going to change. Enums provide an easy way to assign specific values to variables, not letting to set a value outside of the defined ones.

Explain the difference between a `class` and a `struct`.

In summary, the main differences between classes and structures in C# are inheritance (class can, struct cannot), reference type (class) vs value type (struct), default constructor, initialization, and size/performance (struct smaller, generally faster).
Classes are usually used for larger, more complex objects, while structures are used for smaller, simpler objects that are used frequently and need to be passed around quickly. However, both classes and
structures have their own strengths and weaknesses, and the choice between them ultimately depends
on the specific requirements of the project.

A class is a user-defined blueprint or prototype from which objects are created.
Basically, a class combines the fields and methods(member function which define actions) into a single unit.

A structure is a collection of variables of different data types under a single unit.
It is almost similar to a class because both are user-defined data types and both hold a bunch of different data types.

Explain the difference between a `class` and a `record`.

The main difference between class and record type in C# is that a record has the main purpose of storing data, while a class defines responsibility. Records are immutable, while classes are not. Simply put, a class is an OOP concept that wraps data with functionality, while a record represents a set of data.

What are `interfaces`? Why should we use them?

An interfaces defines a contract. Any class or struct that implements the interface must implement all
the members defined in the interfaces. By using interfaces, certain behaviour can be enforced for classes that implement them without the need for inheritance. Classes may implement multiple interfaces. It is good practice to make classes dependent of interfaces rather than implementations of interfaces, so that the actual implementation may be changed without changing that class.

In this regard interfaces make easier to conform to both the Open/Close and the encapsulation principles.

What is `inheritance`?

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. The child class *can* reuse, extend or modify the attributes and behaviour inherited. A class that inherits from another class is called a **derived class**, the class from which other classes inherit
is called **base class**.

Is `multiple inheritance allowed` in C#?

No, the classes may only inherit from a single parent class.

What is a `static class`?

A static class is the same as a non-static class, but a static class cannot be instantiated. Data members of
static classes are accessed via the class name and not the variable name of an instance of a class. A static
class can be used as a convenient container for sets of methods that just operate on input parameters and
do not have to get or set any internal instance fields. For example: System.Math class.

What are the `dangers of using static classes`? How can we `avoid` them?

Dangers:

Some characteristics of using static methods are the following:

- they cannot implement an interface,
- do not support polymorphism,
- cannot have instance members,
- no constructor,
- implicitly abstract,
- methods are slightly faster than methods of non-static classes.

Can easily violate the open-close principle, when a method would need a change at one point.

A utility class filled up with multiple methods can break the single responsibility principle.

Avoid:

`Dependency Injection`: Use dependency injection to provide objects or data rather than relying on static elements.

`Interfaces and Abstraction`: Favor interfaces and abstractions for flexibility and easier testing.

`Singleton Pattern`: Implement the Singleton pattern for controlled instance access without static classes.

`Instance Classes`: Prefer instance classes over static ones for improved maintainability and testability.

What are *extension methods*?

An extension method is a feature in C# that lets us add functionality to a class, struct, enum, interface
or record without modifying their own code. They can be invoked as if they were actually a part of the
class/type they are extending. The `this` keyword must be used in the method's argument. For example
extending the Random class with an overloaded `NextDouble` method looks like this:

```
public static RandomExtension()
{
    public static double NextDouble(this Random random, double min, double max)
    {
        return min + random.NextDouble() * (max - min);
    }
}
```

We will be able to use this method on an instance of the Random class like this:

```
Random myRandom = new Random();
double myDouble = myRandom.NextDouble(10, 20);
//a random double between 10 and 20 will be generated whereas the normal
"NextDouble" method
//takes no parameters and returns a number between 0.0 and 1.0
```

What does the `virtual` keyword means in C#?

Using the `virtual` keyword for a method, property, indexer or event declaration allows us to override it
in a derived class. The implementation of a virtual member can be changed by an overriding member in a
derived class. A non-virtual method cannot be overridden.

What is *overloading* in C#?

Overloading refers to method overloading, which allows us to have methods with the same name, but a different method signature.

A signature is the number of, the order of and the types of arguments that the method takes.

This way we can have methods with the same name that can take different types of arguments and still
perform a similar method on them.

What are `nullable value types`?

Nullability refers to the ability of a type to take `null` as a value. For value types, the type may take on
all values of its underlying value type and an additional `null` value. To make a type nullable, we use the
question mark `?` after the type name: `bool? myBool`. `myBool` can have the values of `true`, `false` and `null`.

What is the `IDisposable` interface used for?

The primary use of the `IDisposable` interface is to release unmanaged resources. Then the Garbage Collector
releases the memory allocated to that object when it is no longer used. There are two ways to indicate that the object is no longer used.

One is to use a `using` statement, the other is invoking the Disposable.Dispose method.

What does the `using` keyword do? When would you use it?

The keyword has two uses.

The first is called the `using directive` which imports types defined in other namespaces.

The second is the `using statement` which defines a scope at the end of which an object (that implements the `IDisposable` interface) is disposed. For example:

```
var numbers = new List<int>();
using (StreamReader reader = File.OpenText("numbers.txt"))
{
    string line;
    while ((line = reader.ReadLine()) is not null)
    {
        if (int.TryParse(line, out int number))
        {
            numbers.Add(number);
        }
    }
}
//StreamReader is disposed here (even if an exception occurs)
```

What is the difference between a using block and a using statement?

The `using` statement ensures the correct use of an IDisposable instance. See example above. When the control leaves the block of the using statement, an acquired IDisposable instance is disposed.

It's also possible to use `using declaration` that doesn't require braces:

```
static IEnumerable<int> LoadNumbers(string filePath)
{
    using StreamReader reader = File.OpenText(filePath);

    var numbers = new List<int>();
    string line;
    while ((line = reader.ReadLine()) is not null)
    {
        if (int.TryParse(line, out int number))
        {
            numbers.Add(number);
        }
    }
    return numbers;
}
```

When declared in a using declaration, a local variable is disposed at the end of the scope in which it's declared. In the preceding example, disposal happens at the end of a method.

How is it possible to use functions as objects in C#?

With the use of "delegates".
Delegate variables are first-class objects and can be assigned to any matching method, or to the value null.
They store a method and its receiver without any parameters:

```
delegate void Notifier(string sender);  // Normal method signature with the
keyword delegate

Notifier greetMe;                       // Delegate variable

void HowAreYou(string sender) {
    Console.WriteLine("How are you, " + sender + '?');
}

greetMe = new Notifier(HowAreYou);

greetMe("Anton");                       // Calls HowAreYou("Anton") and prints
"How are you, Anton?"
```

Also see in the answer below.

What is a `delegate`?

A delegate is a form of type-safe function pointer used by the Common Language Infrastructure (CLI). They specify a method and (optionally) an object to call the method on. They are used to implement callbacks and event listeners.

A delegate object encapsulates a reference to a method.

In praxis it's possible to declare a function and then assign it to a variable then pass it around as any other
type. An example:

```
private int Square(int x)
{
    return x * x;
}

Func<int, int> square = Square;

// "square" is a delegate of type Func<int,int> that encapsulates a function.
//We can now invoke the Square function through the square delegate like this:
int squareOfTwo = square(2);
```

Describe the `Func<TResult>` delegate.

Encapsulates a method that has no parameters and returns a value of the type specified by the TResult parameter.

Describe the `Action` delegate.

Action is a delegate type defined in the System namespace. An Action type delegate is the same as Func
delegate except that the Action delegate doesn't return a value. In other words, an Action delegate can be used with a method that has a void return type.

Example:

```
//with delegate
public delegate void Print(int val);

static void ConsolePrint(int i)
{
    Console.WriteLine(i);
}

static void Main(string[] args)
{
    Print prnt = ConsolePrint;
    prnt(10);
}

//with action
static void ConsolePrint(int i)
{
    Console.WriteLine(i);
}

static void Main(string[] args)
{
    Action<int> printActionDel = ConsolePrint;
    printActionDel(10);
}
```

What does the `abstract` keyword mean in C#?

The abstract keyword enables to create classes and class members that are incomplete and must be implemented in a derived class.

The abstract modifier indicates that the thing being modified has a missing or incomplete implementation. The abstract modifier can be used with classes, methods, properties, indexers,
and events.

What is an `abstract` class?

The abstract modifier in a class declaration indicates that a class is intended only to be a base class of other classes, not instantiated on its own.

Members marked as abstract must be
implemented by non-abstract classes that derive from the abstract class.

Architecture

Explain the `Single Responsibility Principle`.

The Single Responsibility Principle (SRP) dictates that each module, class, or function should have only one specific responsibility or task. This leads to clearer, more maintainable code by minimizing the impact of changes and reducing complexity.

Explain the `Interface Segregation Principle`.

The Interface Segregation Principle (ISP) emphasizes that clients should not be forced to depend on interfaces they do not use entirely. It suggests breaking down large interfaces into smaller, specific ones to avoid unnecessary dependencies. This allows clients to use only the methods they require, promoting flexibility and preventing the implementation of unnecessary functionalities.

What is *`composition over inheritance`*?

Composition over inheritance is oops concept that states classes should achieve polymorphic behavior
and reuse classes using composition. In composition one of the classes has multiple instances of other classes.
In composition the composite `has a` component relation. Prefer composition over inheritance as it is more
malleable / easy to modify later.

Composition: "has a" type of relationship. A car "has an" engine, a person "has a" name, etc.

Inheritance: "is a" type of relationship. A car "is a" vehicle, a person "is a" mammal, etc.

What is a `model class`?

Model classes are a fundamental part of the Model-View-Controller (MVC) architecture. They are used to define data structure, behaviour and interactions of a modelled entity in a software system.

What is a `service class`?

A Service class/interface provides a way of a client to interact with some functionality in the application.
This is typically public, with some business meaning. For example, a TicketingService interface might

allow you to buyTicket, sellTicket and so on.

Explain the `Open/Closed principle`.

The open-closed principle states that software entities should be open for extension, but closed for modification.
This implies that such entities – classes, functions, and so on – should be created in a way that their
core functionalities can be extended to other entities without altering the initial entity's source code.

Explain the `Liskov Substitution Principle`.

The Liskov substitution principle implies that when an instance of a class is passed/extended to another class, the inheriting class should have a use case for all the properties and behavior of the inherited class.

When a class is extended, if some of the properties of the initial class are not useful for the new class,
the Liskov substitution principle is violated. This means that inheritance should not happen here.

`Easier explained`: Replacing the parent class with its child class in the code should be possible without breaking the code.

Explain the `Dependency Inversion Principle`.

High-level modules should not import anything from low-level modules. Both should depend on abstractions (e.g., interfaces).

Abstractions should not depend on details. Details (concrete implementations) should depend on abstractions.

What do we mean by the `Gang of Four (GoF) Design Patterns`?

Design Patterns is a book by the following four people: Erich Gamma, Richar Helm, Ralph Johnson, John Vlissides. They collected 23 software design patterns into this book. They can be groped into 3 categories:

- `Creational patterns`:
    - Abstract factory: groups object factories that have a common theme.
    - Builder constructs: complex objects by separating construction and representation.
    - **Factory method:** creates objects without specifying the exact class to create.
    - **Prototype:** creates objects by cloning an existing object.
    - **Singleton:** restricts object creation for a class to only one instance.
- `Structural`:
    - Adapter: allows classes with incompatible interfaces to work together by wrapping its own interface around that of an already existing class.
    - Bridge: decouples an abstraction from its implementation so that the two can vary independently.
    - Composite: composes zero-or-more similar objects so that they can be manipulated as one object.
    - Decorator: dynamically adds/overrides behaviour in an existing method of an object.
    - Facade: provides a simplified interface to a large body of code.
    - Flyweight: reduces the cost of creating and manipulating a large number of similar objects.
    - Proxy: provides a placeholder for another object to control access, reduce cost, and reduce complexity.
- `Behavioral`
    - Chain of responsibility: delegates commands to a chain of processing objects.
    - Command: creates objects that encapsulate actions and parameters.
    - Interpreter: implements a specialized language.
    - Iterator: accesses the elements of an object sequentially without exposing its underlying representation.
    - Mediator: allows loose coupling between classes by being the only class that has detailed knowledge of their methods.
    - Memento: provides the ability to restore an object to its previous state (undo).
    - **Observer:** is a publish/subscribe pattern, which allows a number of observer objects to see an event.
    - State: allows an object to alter its behavior when its internal state changes.
    - Strategy: allows one of a family of algorithms to be selected on-the-fly at runtime.
    - Template: method defines the skeleton of an algorithm as an abstract class, allowing its subclasses to provide concrete behavior.

What are the `risks associated with using the GoF design patterns`?

The reason these patterns were invented was in many cases related to the deficiencies of programming languages at that time.

As programming languages, and computational capacity have evolved, some of the design pattern
were used less and less frequently, if not even became completely obsolete.

No design pattern will ever replace genuine thinking and problem solving. Many beginner developers,
armed with the knowledge of design patterns will go and try to fit a pattern, or sometimes even multiple patterns to every problem they face. An important thing to note is that every pattern introduces additional complexity to the application, so there is a cost associated with their usage. As such, there should always be a cost-benefit analysis attached to the decision of introducing a pattern.

This is the key to design patterns: to know when to use or not to use them. It's very easy to fall into the trap of overengineering with them.

What do we mean by `YAGNI`?

`You Aren't Gonna Need It.`

It is a principle which arose from extreme programming (XP) that states a programmer should not add functionality until deemed necessary.

Ron Jeffries, a co-founder of XP, explained the philosophy:

```
"Always implement things when you actually need them, never when you just foresee
that you will need them."
```

John Carmack wrote:

```
"It is hard for less experienced developers to appreciate how rarely architecting
for future
requirements / applications turns out net-positive."
```

What do we mean by `SLAP`?

`Single Level of Abstraction Principle` says that every method inside your codebase should deal with
concepts related to just one level of abstraction.

For example, if a method builds and executes HTTP requests, it shouldn't also process the resulting JSON.
These are different abstraction levels: one is about communication over HTTP and the other is about specific data format and layout.

What do we mean by `KISS`?

`Keep it simple, stupid.`

The KISS principle states that most systems work best if they are kept simple rather than made complicated;
therefore, simplicity should be a key goal in design, and unnecessary complexity should be avoided.

What is the `Repository Pattern`?

The `Repository Pattern` is a `design pattern` commonly used in software development to create an `abstraction layer between the application's business logic` and the `data access logic`. It *centralizes data access* and *retrieval operations*, providing a more structured way to work with data. The repository *acts as a mediator between the application* and the *data source*, offering a set of methods to perform **CRUD (Create, Read, Update, Delete) operations** on data entities. This pattern abstracts the underlying data storage, **allowing the application to work with a consistent interface for data access**, **enhancing maintainability**, and **testability** while also promoting separation of concerns within the codebase.

What is a `CRUD interface`?

CRUD apps are the user interface that we use to interact with databases through APIs.
It is a specific type of application that supports the four basic operations: Create, read, update, delete.
A CRUD interface should be able to handle these operations.

Unit testing

Why is `unit testing` a good practice?

Because while writing the code, it is hard to think about all the edge cases, but while writing the test
the approach should be to break the code and that is most easily done with edge cases. Breaking and fixing the code makes the developer more confident in the correct functioning of the code while making the code more robust.

After the code is ready, good unit tests are there to help refactor the code and provide feedback on changes.

What is `Nunit`?

NUnit is an open-source unit-testing framework for all .Net Framework.

What is a `parameterized test`?

A parametrized test is a test that has variables in it, so the base test method can be reused and only
the parameter set must be written out in detail.

What `options` do you have in NUnit to create parameterized tests?

Either using TestCase, TestCaseSource, ValueSource or TestFixture attributes.

What is *`mocking`*?

Mocking is a process used in unit testing when the unit being tested has external dependencies. The purpose of mocking is to isolate and focus on the code being tested and not on the behavior or state of external dependencies.

What is the difference between *mocking*, *stubbing* and *faking*?

Mocking:

`Definition`: Mocking is the process of creating simulated objects (mock objects) that replicate the behavior of real objects or components. It involves imitating the behavior of specific parts of the code to isolate and test their interactions.

`Usage`: Mocks are used for verifying interactions between the code under test and external dependencies.

`Focus`: Primarily used in testing to observe how code interacts with external components without involving the actual implementations of those components.

Stubbing:

`Definition`: Stubbing is a technique used in testing to create simple, predefined objects that provide specific, predetermined responses to method calls. Stubs are less dynamic than mocks and focus on returning hard-coded values or predefined responses.

`Usage`: Stubs are used to provide canned responses to specific calls made during testing without verifying interactions.

`Focus`: Primarily used to test method behavior or specific scenarios without verifying interactions or the flow between different parts of the code.

Faking:

`Definition`: Faking involves creating lightweight, simplified implementations of components or services for testing purposes. Fakes usually have working implementations but with simplified behavior compared to their production counterparts.

`Usage`: Fakes are used when real dependencies are not suitable or practical for testing. For example, using an in-memory database instead of a real database for testing.

`Focus`: Provides simplified implementations of components or services to facilitate testing and improve performance without the complexity of the real implementations.

Databases

What are `relational databases`? What are their `advantages` and `disadvantages`?

Relational databases are databases that make use of relational mathematics. A relational database is a collection of information that organizes data in predefined relationships where data is stored in one of more tables ("relations") of columns and rows.

Relationships are logical connection between different tables.

A relational database (RDB) is a way of structuring information in tables, rows, and columns.
An RDB has the ability to establish links — or relationships – between information by joining tables,
which makes it easy to understand and gain insights about the relationship between various data points.

Advantages:

- categorized data,
- accuracy (no data duplication),
- ease of use,
- collaboration,
- security.

Disadvantages:

- rigid structure that needs planning,
- maintenance issues,
- inflexibility,
- lack of scalability.

How do you `associate entities to each other` in a relational database model?

In a relational database model, entities are associated with each other using relationships. Relationships between entities are established by defining foreign keys in the tables that represent those entities.

What are `tables` in a relational database?

Tables are database objects that contain all the data in a database. In tables, data is logically organized in a row-and-column format similar to a spreadsheet. Each row represents a unique record,
and each column represents a field in the record.

What is a *primary key*?

A primary key is a special relational database table column (or combination of columns) designated
to uniquely identify each table record. A table cannot have more than one primary key. It must be unique for each row of data, cannot be null, every row must have a primary key value.

What is a *foreign key*?

A foreign key is a column in one table that refers to the primary key of another table.
The foreign key establishes a link between the two tables, indicating that the data in the foreign key column of one table corresponds to the data in the primary key column of another table.

What does the `SQL abbreviation` stand for?

Structured Query Language.

What are some of the `SQL database providers` that you've heard of?

MySQL, Oracle, PostgreSQL, Microsoft SQL Server.

Cloud: Amazon Web Services, Oracle, Google, IBM...

What are `SQL data types`? Are there any differences in data types between different SQL databases?

SQL data types define the type of data that can be stored in a database column. Different SQL data types are used to represent different types of data.

Data types can have different names in different databases. Even with the same name, some details might be different. Some databases have their own specific data types.

What are *constraints* in SQL?

Constraints are rules that are used ensure that only data that conforms these rules might be added to
a table. Examples: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK (+ condition), DEFAULT (def value on insert).

How can we `program different SQL databases in C#`?

All or most SQL databases use similar logic. They require a connection string, then a connection object.
Then a connection is opened and a command object is created and used to execute SQL statements against a database. Then the result needs to be processed.

Which SQL statement is used to `create tables`? Describe the syntax briefly.

The CREATE TABLE statement is used to create tables in an SQL database.

```
CREATE TABLE table_name (
   column1 datatype [constraint],
   column2 datatype [constraint],
   ...
   columnN datatype [constraint]
);
```

Which SQL statement can be used to `insert values`? Describe the syntax briefly.

The INSERT statement.

```
INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2,
value3);
```

Which SQL statement can be used to `update values`? Describe the syntax briefly.

The UPDATE statement.

```
UPDATE table_name SET field1=value1, field2=value2 WHERE id=currentid;
```

Which SQL statement can be used to `delete rows`?. Describe the syntax briefly.

The DELETE statement.

```
DELETE FROM table_name WHERE id=currentid;
```

Which SQL statement can be used to create queries?. Describe the syntax briefly.

The SELECT statement.

```
SELECT column1, column2, column3 FROM table_name;
```

How can you `join tables together` in SQL? When should you do it?

With one of the `JOIN statements (LEFT, RIGHT, INNER, FULL OUTER)`.

It should be done when the required needs to be combined from multiple tables.

The common columns between these tables should be identified and used to perform the join.

Advanced C#

Web development

What is an `API`?

API stands for **Application Programming Interface.** It is a set of rules and protocols that allows different software applications to communicate with each other. APIs define the methods and data formats that developers can use to request and exchange information between systems.

What is a `REST`?

REST stands for **Representational State Transfer.** It is an architectural style for designing networked applications and a set of constraints that developers can apply when creating web services or APIs. REST is not a technology or protocol itself but rather a set of principles for building scalable and efficient web services

Describe the `HTTP protocol and its methods`.

```
- GET: Retrieve data from the server without making any changes. It is used for
reading resources.
- POST: Submit data to the server to create a new resource. It is used for
creating data on the server.
- PUT: Update an existing resource on the server. It is used for making changes
to a resource.
- DELETE: Remove a resource from the server. It is used to delete data.
- PATCH: Partially update a resource on the server. It is used for making partial
changes to a resource.
- HEAD: Retrieve only the headers of a resource, without the actual data. It is
often used for checking if a resource has been modified.
```

How do you identify `unique resources on the web`?

Unique resources on the web are typically identified using `Uniform Resource Locators (URLs)`, which are also known as `web addresses`. A URL is a **string of characters that provides a reference to a specific resource on the internet**. URLs are used to identify and access various types of resources, including web pages, images, documents, and web services. A URL consists of several components that collectively specify the location and identity of the resource.

What are the `main frontend languages` on the web? Describe them briefly.

1. **HTML (Hypertext Markup Language):**
    - HTML is the backbone of web content and structure.
    - It provides the basic markup and elements to define the structure of web pages, such as headings, paragraphs, links, images, forms, and more.
    - HTML5 is the latest version, offering enhanced multimedia and interactive features.
2. **CSS (Cascading Style Sheets):**
    - CSS is used to style and format the HTML content, defining the layout, colors, fonts, and visual presentation of web pages.
    - It allows for the separation of content (HTML) and presentation (CSS), making it easier to maintain and style web pages.
    - CSS3 introduced advanced features like animations and transitions.
3. **JavaScript:**
    - JavaScript is a versatile, high-level programming language used to add interactivity and dynamic behavior to web pages.
    - It can be used for client-side scripting to create interactive forms, validate user input, and handle events, among other tasks.
    - JavaScript is essential for building modern web applications and is supported by all major web browsers.
4. **TypeScript:**
    - TypeScript is a superset of JavaScript that adds static typing to the language.
    - It helps catch errors during development and provides better tooling support for large codebases.
    - TypeScript is often used in larger, enterprise-level web applications.
5. **React:**
    - React is a JavaScript library for building user interfaces.
    - It is maintained by Facebook and is known for its component-based architecture, which makes it easy to create reusable UI components.
    - React is often used in single-page applications (SPAs) and is a popular choice for modern web development.
6. **Vue.js:**
    - Vue.js is another JavaScript framework for building user interfaces.
    - It is known for its simplicity and flexibility and offers a progressive framework that can be incrementally adopted in existing projects.
    - Vue.js is gaining popularity for web development.

7. **Angular:**
    - Angular is a comprehensive TypeScript-based framework for building web applications.
    - It provides a full-featured platform for building large-scale, enterprise-level applications.
    - Angular has a strong focus on modularity and testability.
8. **Sass and Less:**
    - Sass (Syntactically Awesome Style Sheets) and Less are CSS preprocessors.
    - They extend the capabilities of CSS by adding features like variables, nesting, and functions, making it easier to write and maintain complex styles.
9. **Bootstrap:**
    - Bootstrap is a popular CSS framework that provides pre-designed, responsive UI components and a grid system.
    - It allows for rapid development of visually appealing and mobile-friendly web applications.
10. **JQuery:**
    - jQuery is a JavaScript library that simplifies DOM manipulation and event handling.
    - While its usage has declined with the rise of modern JavaScript frameworks, it is still found in legacy applications.

What is JSON?

JSON stands for **JavaScript Object Notation**. It is a `lightweight data interchange format` that is **easy for humans to read and write** and **easy for machines to parse and generate**. JSON is often used to **transmit data between a server and a web application**, as well as to **store configuration settings and data** for various applications.

What `operations` are `fundamental` to most `data-driven applications`?

`Data-driven applications` typically involve a range of fundamental operations to effectively **collect, process, analyze, and utilize data**. These operations can vary depending on the specific application and its goals, but some common fundamental operations include:

1. **Data Collection**:
    - Data Ingestion: The process of collecting data from various sources, which may include databases, APIs, sensors, logs, and external data feeds.
    - Data Integration: Combining and transforming data from different sources to create a unified dataset.
2. **Data Storage**:
    - Data Warehousing: Storing structured data in a centralized repository optimized for query and analysis.
    - Data Lakes: Storing both structured and unstructured data in a flexible, scalable repository.
3. **Data Processing**:
    - Data Transformation: Converting data into a suitable format for analysis, often involving cleaning, normalization, and enrichment.
    - Data Aggregation: Summarizing and grouping data for reporting or analysis.
    - Data ETL (Extract, Transform, Load): Automating the extraction, transformation, and loading of data from source systems to the target data store.
4. **Data Analysis**:
    - Data Exploration: Examining data to identify patterns, trends, and anomalies.
    - Data Modeling: Creating statistical, machine learning, or other models to make predictions, classifications, or recommendations.
    - Data Visualization: Representing data in graphs, charts, and dashboards to facilitate understanding and decision-making.
5. **Data Storage Management**:
    - Data Security: Implementing measures to protect data from unauthorized access and breaches.
    - Data Backup and Recovery: Ensuring data can be restored in case of data loss or system failures.
    - Data Archiving: Moving less frequently accessed data to long-term storage to optimize performance and cost.
6. **Data Governance**:
    - Data Quality Assurance: Ensuring data is accurate, consistent, and reliable.
    - Data Privacy and Compliance: Adhering to data protection regulations and privacy policies.
    - Data Cataloging: Creating metadata and documentation to make data assets discoverable and understandable.

7. **Data Integration and Real-Time Processing**:
    - Real-Time Data Streaming: Processing and analyzing data as it is generated, allowing for immediate responses and insights.
    - Integration with External Systems: Connecting to other software systems or services for data exchange.
8. **Machine Learning and AI**:
    - Training and Deployment: Developing machine learning models and deploying them to make predictions or automate tasks.
    - Model Monitoring: Continuously assessing model performance and retraining models as needed.
9. **Reporting and Decision Support**:
    - Dashboard Creation: Building interactive dashboards and reports for stakeholders to monitor and make decisions based on data.
    - Alerting and Notifications: Setting up automated alerts for specific data events or conditions.
10. **Data Access and API**:
    - Data APIs: Providing interfaces for other applications to access and interact with the data.
    - User Access Control: Managing permissions and access levels for users and applications.
11. **Feedback Loop**:
    - Collecting user feedback and system performance data to iterate and improve the application.

Which web protocol would you use if you need a `bi-directional communication channel`?

If you need a `bi-directional communication channel` for web applications, one of the commonly used web protocols is **WebSocket**. **WebSocket** is designed to **provide full-duplex, bidirectional communication** between a `client` (usually a web browser) and a `server` over a single, long-lived connection.
Unlike the traditional request-response cycle of HTTP, WebSocket allows for real-time, low-latency, and continuous data exchange between the client and server.

ASP.NET Core

Describe the different `application types` available in `ASP.NET Core`.

1. **MVC (Model-View-Controller):**
    - MVC is a design pattern and architectural approach for building web applications.
    - ASP.NET Core MVC provides a framework for creating web applications structured around the model (data), view (presentation), and controller (logic) components.
    - It is suitable for building traditional web applications where you need fine-grained control over the structure and behavior.
2. **Razor Pages:**
    - Razor Pages is a lightweight web application framework that simplifies building web pages for simpler scenarios.
    - It is well-suited for developing small to medium-sized applications where the focus is on individual web pages rather than a complex application structure.
    - Razor Pages combine HTML markup with C# code using the Razor view engine.
3. **Web API:**
    - ASP.NET Core is excellent for building RESTful APIs that serve data to client applications, including web and mobile apps.
    - ASP.NET Core Web API provides a framework for creating APIs with built-in support for content negotiation, request/response serialization, and routing.
4. **SignalR:**
    - SignalR is a library for adding real-time functionality to web applications, enabling server-to-client and client-to-client communication.
    - It is used for building features like chat applications, online gaming, live notifications, and collaborative tools.
5. **Blazor:**
    - Blazor is a web framework that allows you to build interactive web applications using C# and .NET instead of JavaScript.
    - It offers both client-side (WebAssembly) and server-side (Razor Components) hosting models, allowing you to choose between running your C# code in the browser or on the server.
6. **GraphQL:**
    - ASP.NET Core can be used to build GraphQL APIs, which provide a more flexible and efficient alternative to REST for querying and manipulating data.
    - It allows clients to request exactly the data they need, reducing over-fetching and under-fetching of data.

7. **Razor Class Libraries:**
    - Razor Class Libraries (RCLs) are reusable libraries that contain Razor views, pages, and components.
    - They can be shared across multiple ASP.NET Core applications, making it easier to encapsulate UI components and reuse them.
8. **Console Applications:**
    - ASP.NET Core also supports building command-line applications, making it suitable for creating background tasks, utilities, or automation scripts.
9. **Azure Functions:**
    - You can build serverless applications using Azure Functions with ASP.NET Core.
    - Azure Functions are event-driven and can be used for various purposes, such as processing data, running background tasks, and handling HTTP requests.
10. **Microservices:**
    - ASP.NET Core can be used to build microservices, which are small, independently deployable services that work together to form a larger application.
    - Microservices architecture is well-supported in ASP.NET Core, and you can use technologies like Docker and Kubernetes for containerization and orchestration.

Which `application type` should you choose if you want to implement a `REST API`?

If you want to implement a REST API in ASP.NET Core, you should choose the **Web API** application type. **ASP.NET Core Web API** is specifically designed for building `RESTful` web services and APIs.

Explain the concept of `middleware` in ASP.NET Core and give an `example of how you can use it` in a web application.

In ASP.NET Core, `middleware` is a **key component of the request processing pipeline**. `Middleware components` are responsible for **processing** and potentially **modifying HTTP requests and responses** as they flow through the pipeline.
They provide a way to add various behaviors and functionality to your web application, such as authentication, routing, logging, error handling, and more, in a modular and configurable manner.

`Middleware components` are executed in a specific order, from the first middleware in the pipeline to the last. Each middleware component can choose to pass the request further down the pipeline to the next middleware, terminate the pipeline, or short-circuit the request/response process.

```
public class CustomHeaderMiddleware
{
    private readonly RequestDelegate _next;

    public CustomHeaderMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        // Add a custom response header
        context.Response.Headers.Add("X-Custom-Header", "Hello from
CustomHeaderMiddleware");

        // Call the next middleware in the pipeline
        await _next(context);
    }
}

public static class CustomHeaderMiddlewareExtensions
{
    public static IApplicationBuilder UseCustomHeaderMiddleware(this
IApplicationBuilder builder)
    {
        return builder.UseMiddleware<CustomHeaderMiddleware>();
    }
}
```

In this example:

1. We define a custom middleware component called `CustomHeaderMiddleware`, which adds a custom response header.
2. The `InvokeAsync` method is called for each request. It adds the custom header and then calls the next middleware in the pipeline using `_next(context)`.
3. We also define an extension method `UseCustomHeaderMiddleware` to make it easier to use the middleware in the `Startup.cs` file.

What is a `Controller` class? Describe its function.

In the context of ASP.NET Core, a `Controller class` is a **fundamental component** used to **handle incoming HTTP requests and generate responses**. Controllers play a **central role** in the `Model-View-Controller (MVC)` architectural pattern and are **crucial** for building web applications.

Its functions are: `Request Handling`, `Action Methods`, `Data Processing`, `Response Generation`, `View Rendering`

What is a `Controller Action Method`?

A controller action method is a C# method within a controller class that **handles a specific HTTP request and returns a response**.

How do you configure `routing` in ASP.NET Core, and what role does it play in `handling incoming requests`?

Configuring Routing:

1. **Route Configuration in Startup**:
   - In an ASP.NET Core application, routing is configured in the `Startup.cs` file. The `Configure` method is where you define routes.
   - The `Configure` method uses the `UseEndpoints` extension method to define routes, specifying the routing pattern and the associated controller and action.
2. **Route Templates**:
   - Route templates define URL patterns and specify placeholders for route parameters. For example, `"/products/{id}"` is a route template where `{id}` is a placeholder for a parameter.
3. **Controller Attributes**:
   - Controllers and action methods can be annotated with route attributes to further define routing. For example, the `[Route]` attribute can be applied to a controller or action to specify a route pattern for that specific controller or action.
4. **Default Routes**:
   - You can configure default routes, which define a default controller and action to be used when no specific route matches the request.

Role of Routing in Handling Requests:

1. **Request Routing**:
   - Routing plays a crucial role in determining which controller and action method should be invoked to handle an incoming HTTP request.
   - When a client makes a request to a specific URL, ASP.NET Core uses the routing configuration to match the request to a route template and identify the associated controller and action.
2. **Parameter Extraction**:
   - Route templates can contain placeholders for parameters, such as `{id}`. When a URL matches a route template, ASP.NET Core extracts values from the URL and maps them to the corresponding action method parameters.
3. **Controller Selection**:
   - Routing determines which controller is responsible for handling a request based on the URL and route configuration. For example, if a URL matches the route for the "Products" controller, the "Products" controller is selected.
4. **Action Method Invocation**:
   - Routing also selects the appropriate action method within the chosen controller to handle the request. This is based on the HTTP verb (GET, POST, etc.) and the action method name.
5. **Custom Routes**: Custom route attributes applied to controllers and action methods allow for more fine-grained control over routing, enabling you to create SEO-friendly URLs and handle complex routing scenarios.

6. **Default Routes**:
    - Default routes provide a fallback mechanism for handling requests when no specific route template matches. They ensure that requests are directed to a default controller and action.

What options do you have for `handling errors` in ASP.NET Core?

ASP.NET Core provides several options for handling errors and exceptions. The choice of error handling mechanism *depends on the nature of the error*, the desired behavior, and the specific requirements of your application.
Examples: `Middleware-Based Exception Handling`, `Use of try...catch Blocks`, `Custom Exception Filters`, `Global Exception Filters`, `Use of the [ExceptionHandler] Attribute`, `HTTP Error Status Codes`, `Error Pages`

What are the `advantages` of using `dependency injection` in ASP.NET Core? How does it help with `testing and maintainability`?

`Dependency injection` in ASP.NET Core promotes **maintainable and testable code** by **decoupling components**, allowing for the **substitution of dependencies**, and facilitating the creation of modular, **reusable**, and **flexible applications**. It is a **fundamental aspect** of modern software engineering practices and contributes to the long-term sustainability of your applications.

What is ASP.NET Core's `appsettings.json` file, and how is it used to manage `configuration settings` in a web application?

ASP.NET Core's `appsettings.json` file is a configuration file used to manage various configuration settings for a web application. It is part of the ASP.NET Core configuration system and plays a key role in **separating configuration data from the application code**. This approach makes it **easier to configure and maintain an application** for different environments, such as development, staging, and production.

How can you implement `authentication` and `authorization` in ASP.NET Core? Mention some of the `authentication providers` available in the framework.

ASP.NET Core includes `authentication middleware` that you can configure in the Startup.cs file.
`UseAuthentication`: Configures authentication services.
`UseAuthorization`: Configures authorization services.
Here are some authentication providers available in the framework: `Cookie Authentication, JWT (JSON Web Token) Authentication, OAuth and OpenID Connect, Windows Authentication`

How can you `unit test` your ASP.NET Core application?

ASP.NET Core supports various testing frameworks such as `xUnit`, `MSTest`, and `Nunit`

Should you consider unit testing your `Controller` classes?

A controller should contain as little behavior as possible. It should instantiate and call domain classes that are unit tested themselves. A controller should only be the orchestrator. **Unit tests should only need to verify that it setup the right domain things**.

What is `integration testing`, and how does it *differ* from `unit testing`?

`Integration testing` is a type of `software testing` that **focuses on evaluating the interactions** and **integration points between different components**, **modules**, or **services** within a software system. It aims to **ensure that these components work correctly together** when combined to form a larger, integrated application.

C# features

How do you `call APIs` from C# code?

APIs can be called from C# code using `HTTP client libraries` like `HttpClient` to make requests to API endpoints.

How can you achieve `Clean Architecture` in a C# application?

Achieving Clean Architecture in a C# application involves designing your application with a set of **principles and patterns** that promote these goals.
Such as: `Layered Architecture, Dependency Inversion Principle (DIP), Interfaces and Abstractions, Single Responsibility Principle (SRP), Test-Driven Development (TDD)`

What is `async/await` in C#?

`async/await` is a feature in C# that allows you to write asynchronous code more efficiently. It simplifies asynchronous programming and **improves responsiveness** in applications.

What is the `TPL` in .NET?

TPL stands for **Task Parallel Library**. TPL is a **set of APIs** in .NET that **simplifies parallel and asynchronous programming**. It includes **tasks, parallel loops, and more**.

Describe the various ways to achieve `concurrency` in a C# program.

Concurrency can be achieved in C# through features like **multi-threading, asynchronous programming, and parallel programming libraries**.

Why is it considered `good practice` to implement `async controller methods`?

Implementing async controller methods in ASP.NET Core **improves application responsiveness** and **scalability** by **avoiding thread blocking during I/O-bound operations**.

Design patterns

Describe the `Repository pattern`. When should you consider implementing it?

The Repository pattern is a design pattern that **separates the logic that retrieves data from a data store (repository) from the business logic in an application**.
Implement the Repository Pattern **when you need a centralized, consistent interface to interact with multiple data sources**, want to separate data access from business logic, and aim for **improved maintainability, testability, and flexibility in handling data operations** within the application.

What is the `Unit of Work pattern`, and what construct is available in C# to use it?

The `Unit of Work pattern` is used to **maintain data consistency** and **transaction control** in applications. `Entity Framework` provides a `DbContext` class to implement this pattern.

What is `Clean Architecture`?

`Clean Architecture` is a **software architectural pattern** that emphasizes separation of concerns, maintainability, and testability by **structuring the application into distinct layers with clear dependencies and responsibilities**.

Entity Relationships, ORM, EF

What is `ORM (Object Relational Mapping)`? What are the `benefits`, and `when to use it`?

`ORM (Object-Relational Mapping)` is a programming technique that **facilitates the conversion of data** between **incompatible type systems** in object-oriented programming languages and **relational databases**. It **maps object-oriented models to relational databases**, allowing the application to **interact with the database** using an object-oriented approach.

Benefits of ORM:

- `Simplification`: Reduces the need for manual data conversion between objects and the database, making development more straightforward.
- `Portability`: Allows easier migration between different database systems without changing the application's code extensively.
- `Productivity`: Speeds up development by providing higher-level abstractions and automated handling of database operations.
- `Maintenance`: Enhances code maintainability by offering a more intuitive way to work with databases in object-oriented code.
  When to Use ORM:
- `Complex Data Relationships`: Dealing with complex relationships between objects and relational databases, where manual data mapping would be time-consuming and error-prone.
- `Rapid Development`: During rapid development cycles, as ORM tools can significantly speed up the process by automating object-to-database mapping.
- `Changing Database Systems`: When there's a need to switch between different database systems or vendors, as ORM provides abstraction, reducing the effort required for changes.
- `Avoiding SQL Queries`: To abstract away the use of direct SQL queries in the application code, enabling developers to work at a higher level of abstraction.
- `Maintainability and Code Clarity`: For better code maintainability and clarity, especially when dealing with object-oriented models rather than direct SQL statements, making the code more readable and easier to maintain.

Explain the concept of `cardinality` in entity relationships. How does it impact the design and mapping of entities?

`Cardinality` in entity relationships **defines the number of instances** of one entity that can be associated with the number of instances of another entity.

- One-to-One (1:1): Each instance in one entity is related to exactly one instance in another entity.
- One-to-Many (1:N): Each instance in one entity can be related to multiple instances in another entity.
- Many-to-One (N:1): Multiple instances in one entity are related to exactly one instance in another entity.
- Many-to-Many (N:N): Multiple instances in one entity are related to multiple instances in another entity.

Impact on Design and Mapping of Entities

- `Database Schema`: Cardinality impacts **how tables are designed** in a database schema, **influencing foreign key relationships**.
- `Application Logic`: Cardinality **determines how data is accessed and manipulated** in the application code, **affecting how relationships are handled**.

Describe the general process of working with Entity Framework Core.

Here are the major steps in working with EF Core:

- Set up your project.
- Install `Entity Framework Core` and `database provider packages`.
- Define your model.
- Create a subclass of `DbContext`.
- Configure the `DbContext`.
- Use migrations for database schema changes.
- Perform database operations using the `DbContext`.
- Handle transactions.
- Implement `error handling` and `validation`.
- Use `asynchronous` programming for **better performance**.
- Write `unit tests`.
- Implement `logging` and `monitoring`.
- Optimize database queries.
- Ensure the target environment has the required database provider.
- Implement `error handling` and `recovery strategies`.
- Integrate `Entity Framework Core` into your `CI/CD pipeline`.

DevOps

What is `Docker` and `Docker Compose`?

`Docker` is a **containerization platform** that allows you to **package applications and their dependencies into containers**.
`Docker Compose` is a tool for **defining and running multi-container** `Docker` **applications**.

Explain the difference between an `image` and a `container` in Docker.

- **Image**: An image is a **lightweight, stand-alone, and executable package** that **includes all the necessary code, runtime, libraries**, and **system tools** required to run a piece of software. Images serve as a **blueprint** for creating containers.
- **Container**: A container is an **instance of an image** that **runs in an isolated environment**. It **encapsulates the application and its dependencies**, ensuring `consistency` and `portability` across different environments. Containers are the **runnable instances of Docker images**.

What is a `Dockerfile`?

A `Dockerfile` is a **text file used to define a set of instructions for building a Docker image**. It serves as a **recipe for creating a reproducible and self-contained image** that contains all the necessary components to run an application.

What is `CI/CD (Continuous Integration/Continuous Deployment)`, and what are its benefits?

`CI/CD` is a set of **practices and tools** for **automating the building, testing, and deployment** of applications. It **improves software quality and delivery speed**.

Describe an example `CI/CD pipeline` for an ASP.NET Core.

1. **Source Code Repository**:
   - The pipeline begins with a source code repository, such as `Git (GitHub, GitLab, Bitbucket)`, where developers push code changes.
2. **Continuous Integration (CI) Stage**:
   - **Source Code Trigger**: Whenever a developer pushes code changes to the repository, a CI server `(e.g., Jenkins, Travis CI, CircleCI, Azure DevOps)` is triggered.
   - **Build**: The CI server starts by pulling the latest code from the repository and initiating a build process. In the case of ASP.NET Core, it runs commands to compile the application using the `dotnet build` command.
   - **Unit Testing**: The CI server runs unit tests `(e.g., xUnit, MSTest, NUnit)` to ensure code quality. Test results are collected and reported.
   - **Code Analysis**: Optionally, static code analysis tools `(e.g., SonarQube, ESLint)` can be used to analyze the code for quality and security issues.
   - **Containerization**: If the application is intended to be deployed in containers, the CI server can build Docker images using a `Dockerfile`.
   - **Artifact Generation**: After a successful build and testing, the CI server generates artifacts, such as `compiled binaries`, `Docker images`, and `configuration files`.
3. **Continuous Deployment (CD) Stage**:
   - **Deployment to Staging Environment**:
     - The CD process deploys the application to a staging environment for further testing and validation.
     - This can include deploying to a cloud platform like `Azure, AWS, or Google Cloud`, or an on-premises server.
     - Database schema updates can be managed with database migration scripts `(e.g., Entity Framework Core Migrations)`.
   - **Integration Testing**: In the staging environment, integration tests are performed to ensure that different parts of the application work together correctly.
   - **Approval Gates**: Manual or automated approval gates can be implemented, where designated team members or automated checks verify that the application is ready for production deployment.
   - **Deployment to Production**: Upon approval, the CD process deploys the application to the production environment. This can be a gradual rollout or a full deployment, depending on the organization's strategy.
   - **Smoke Testing**: Smoke tests are performed to quickly verify that the deployed application is operational.
4. **Monitoring and Observability**:
   - Application performance, logs, and metrics are monitored in the production environment.
   - Tools like Prometheus, Grafana, or Application Insights may be used for monitoring.

5. **Automated Rollback (if necessary)**:
    - If issues are detected in the production environment, automated rollback procedures can revert to the previous stable version.
6. **Reporting and Notifications**:
    - The CI/CD pipeline generates reports and notifications, providing insights into the success and failure of builds and deployments.
    - Notifications can be sent to development teams and stakeholders.
7. **Security Scanning**:
    - Security scans, such as vulnerability scans and penetration testing, may be incorporated into the pipeline to ensure the application's security.
8. **Feedback Loop**:
    - The pipeline collects feedback, such as testing results and performance metrics, to improve code quality and deployment processes.
9. **Version Management**:
    - Version control and management are essential to track changes and roll back to previous versions if needed.
10. **Continuous Documentation**:
    - Keep documentation up to date, including release notes, deployment guides, and infrastructure documentation.
11. **Scalability**:
    - Plan for scalability in terms of infrastructure, automation, and deployment to accommodate increased workloads and growing user bases.

SQL

What are `aggregate functions`? Could you give a few examples?

Aggregate functions in SQL are used to perform calculations on sets of values. Examples include `SUM, AVG, COUNT, MIN, and MAX.`

What does `GROUP BY` do?

The `GROUP BY` clause in `SQL` is used to **group rows that have the same values in specified columns**. It is typically used in combination with aggregate functions.

When would you use `HAVING`?

The `HAVING` clause in `SQL` is used to **filter the results of a `GROUP BY` query** based on the results of aggregate functions applied to groups of rows. You would use the `HAVING` clause when you want to **filter the grouped data based on a condition that involves aggregated values**.

What does a `LEFT OUTER JOIN` do and how does it differ from an `INNER JOIN`? Give an example for a result table for each `JOIN` type.

`LEFT OUTER JOIN`:

- A `LEFT OUTER JOIN`, often simply referred to as a `LEFT JOIN`, **returns all the rows from the left table and the matched rows from the right table**. If there is no match in the right table, NULL values are used for columns from the right table in the result.

Example:

Using the same `Orders` and `Customers` tables, a `LEFT JOIN` would return all the rows from the `Orders` table and any matching rows from the `Customers` table. If there is no matching customer for an order, the `CustomerName` column will contain NULL.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
LEFT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Result Table (simplified):

```
OrderID | CustomerName
------- | ------------
1       | John
2       | Alice
3       | Bob
4       | NULL
```

`INNER JOIN`:

- An `INNER JOIN` **returns only the rows where there is a match in both tables**. If there is no match in one of the tables, the row is not included in the result.

Example:
Suppose we have two tables: `Orders` and `Customers`. The `CustomerID` in the `Orders` table is related to the `CustomerID` in the `Customers` table. An `INNER JOIN` between these tables would return only the rows where there is a match in both tables.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Result Table (simplified):

```
OrderID | CustomerName
------- | ------------
1       | John
2       | Alice
3       | Bob
```

Can you insert multiple records with a single `INSERT` statement?

Yes, you can insert multiple records with a single `INSERT` statement in `SQL`. This is often referred to as a `bulk insert` or `multi-row insert`. To insert multiple records, you can use the `INSERT INTO` statement with a list of values enclosed in parentheses. Each set of values represents a separate row to be inserted.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES
    (value1_1, value1_2, value1_3, ...),
    (value2_1, value2_2, value2_3, ...),
    (value3_1, value3_2, value3_3, ...),
    ...;
```

When do you use the `DISTINCT` keyword?

The `DISTINCT` keyword in `SQL` is used to **eliminate duplicate rows from the result** set of a `SELECT` query.

What would you use the `CASCADE` keyword for?

The `CASCADE` keyword in `SQL` is used in the context of **foreign key constraints and data manipulation** to define the behavior when **rows in a referenced (child) table are affected by changes in the parent (referenced) table**. It specifies that changes in the parent table should be propagated to the child table in certain ways.

What are `database transactions`?

A `database transaction` is a fundamental concept in database management systems `(DBMS)` and is often used to ensure the **consistency, integrity, and reliability of database operations**. A transaction is a **sequence of one or more database operations that are treated as a single, indivisible unit of work**. Database transactions are typically initiated using `SQL` statements such as `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK`.

What is an `inner/nested query`?

An `inner or nested query`, often referred to as a `subquery` or a `subselect`, is a `SQL` **query embedded within another query**.
`The inner query` **is executed first** and its **result is used as a part of the outer query**.
`Subqueries` are used for various purposes, including **filtering, joining, or performing calculations** based on data** from one table** within the context of data **from another table**.

Example:

```
SELECT product_name
FROM products
WHERE price > (SELECT AVG(price) FROM products);
```

What is a `database index`?

A `database index` is a database structure that **enhances the speed of data retrieval operations on a database table**.
It provides a quick and efficient way to **look up records based on the values in one or more columns**.

What entity/table relationship types do you know?

`One-to-One, One-to-Many (or Many-to-One), Many-to-Many`

What do you know about `database normalization`?

`Database normalization` is a database design technique used to **organize data in a relational database management system** `(RDBMS)` efficiently and **reduce data redundancy while preserving data integrity**.
The process of normalization involves **breaking down a complex database into simpler, more manageable** tables and defining relationships between them.
The primary goal of normalization is to **eliminate data anomalies and ensure** that data is stored consistently and without redundancy.

What is `SQL injection`?

`SQL injection` is a malicious technique that **attackers use to exploit vulnerabilities in web applications and gain unauthorized access to a database**.
It occurs **when an application allows users to enter unsanitized input**, which is then used in `SQL` queries **without proper validation or escaping**.
`SQL injection` can lead to a wide range of **security issues**, including **unauthorized data access, data manipulation, and even remote code execution**.