# OOP

class Person

{

// Properties

public string Name { get; set; }

public int Age { get; set; }

// Constructor

public Person(string name, int age)

{

Name = name;

Age = age;

}

// Method

public void SayHello()

{

Console.WriteLine($"Hello, my name is {Name} and I am {Age} years old.");

}

}

* **Abstraction**:
  + is the process of hiding the internal details of an application from the outer world
  + Is used to describe things in simple terms
  + It’s used to create a boundary between the application and the client program

public abstract class Shape {

public abstract double Area(); // Abstract method, no implementation

public void Display() {

Console.WriteLine("This is a shape.");

}

}

* **Encapsulation**:
  + Is the packing of data and functions into a single component. The features of encapsulation are supported using classes in most object-oriented programming languages, although other alternatives also exist.
  + It allows selective (by using access modifier: public, private protected, internal, protected internal) hiding of properties and methods in an object by building an impenetrable wall to protect the code from accidental corruption.
* **Inheritance**:
  + Inheritance is when a class is based on another class.
  + Inheritance enables you to create new classes that reuse, extend, and modify the behavior that is defined in other classes.
  + The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class. A derived class can have only one direct base class.

class Student : Person

{

public string StudentId { get; set; }

public Student(string name, int age, string studentId)

: base(name, age)

{

StudentId = studentId;

}

public void Study()

{

Console.WriteLine($"{Name} is studying with student ID {StudentId}.");

}

}

* **Polymorphism**:
  + Polymorphism is the provision of a single interface to entities of different types.
  + A polymorphic type is a type whose operations can also be applied to values of some other type, or types.

Person person = new Student("Charlie", 20, "12345");

person.SayHello();

# Dependency Injection

* Is a design pattern in which an object or function receives other objects or functions that it depends on.
* A form of inversion of control, dependency injection aim to separate the concerns of constructing objects and using them, leading to loosely coupled programs
* There are basically three types of dependency injection:
  + Constructor injection
  + Setter injection
  + Interface injection
* Implement:
  + Create interface
  + Create class that is inherited interface
  + After that, at program file we can register service for dependency injection

# Dependency injection in software development:

**AddTransient:** Creates a new instance every time it's requested.

**AddScoped:** Creates a single instance per client request.

**AddSingleton:** Creates a single, shared instance for the entire application's lifetime.

# SOLID principles

* **Single Responsibility Principle (SRP):** A class should have only one reason to change.
* **Open/Closed Principle (OCP):** Software entities should be open for extension but closed for modification via Inheritance and Polimophism .
* **Liskov Substitution Principle (LSP):** Subtypes must be substitutable for their base types without altering program correctness.
* **Interface Segregation Principle (ISP):** Clients should not be forced to depend on interfaces they don't use.
* **Dependency Inversion Principle (DIP):** High-level modules should not depend on low-level modules; both should depend on abstractions.

# Can a class extend multiple classes in C#

In C#, a class can only directly inherit from a single base class. This is known as single inheritance. In other words, C# does not support multiple inheritance for classes, where a class can inherit from more than one class. In C#, a class can only directly inherit from a single base class. This is known as single inheritance. In other words, C# does not support multiple inheritance for classes, where a class can inherit from more than one class.

# WCF - Windows Communication Foundation

WCF in C# WinForms offers benefits like:

1. **Interoperability:** WCF can communicate with clients written in different languages.
2. **Service Orientation:** Supports service-oriented architecture (SOA), making the application scalable.
3. **Multiple Message Patterns:** Provides one-way messaging, request-reply, and duplex.
4. **Protocol and Serialization Flexibility:** You can switch protocols and serialization without changing code.
5. **Hosting Flexibility:** WCF services can be hosted in different ways.
6. **Security:** Supports features like authentication, authorization, integrity, confidentiality.

# User Control:

* **Purpose:** Groups existing controls into a single, reusable unit.
* **Design:** Built using the familiar toolbox of standard WinForms controls.
* **Functionality:** Inherits from the UserControl class and provides a familiar design experience within Visual Studio.
* **Use Case:** Ideal for creating reusable sections of UI that you want to use in multiple places within your application. Imagine a custom address input section that you can use on different forms for data collection.

# Custom Control:

* **Purpose:** Offers more granular control over appearance and behavior.
* **Design:** Requires inheriting directly from the Control class and overriding the OnPaint method to handle how the control is drawn.
* **Functionality:** Provides greater flexibility for creating controls with unique visual elements or behaviors that don't fit the mold of standard controls.
* **Use Case:** Suitable for building custom UI components that require specific visual styles or functionality beyond what standard controls provide. An example might be a custom gauge control for displaying data levels.

# Compare User Control and Custom Control

|  |  |  |
| --- | --- | --- |
| **Feature** | **User Control** | **Custom Control** |
| **Base Class** | UserControl | Control |
| **Design Experience** | Familiar toolbox of standard WinForms controls | Requires overriding OnPaint for custom drawing |
| **Functionality** | Reuses existing controls | Offers more granular control |
| **Use Case** | Reusable UI sections | Unique visual elements or behaviors |

# Usually use control

In C# WinForms (Windows Forms), there are several commonly used controls that help in creating interactive user interfaces. Here are some of them:

1. **Button**: This is a clickable control that can trigger an event when clicked.
2. **TextBox**: This control allows users to input text.
3. **Label**: This is used to display static text on the form.
4. **CheckBox**: This allows users to select or deselect an option.
5. **RadioButton**: This is used when you want the user to select only one option from a group of options.
6. **ComboBox**: This is a drop-down list that allows users to select an option from a list.
7. **ListBox**: This displays a list of items that users can select from.
8. **DataGridView**: This is a highly flexible grid that can display data from a database or other data source.
9. **PictureBox**: This control is used to display images.
10. **MenuStrip**: This is used to create menus in the application.
11. **ProgressBar**: This control visually indicates the progress of a lengthy operation.
12. **DateTimePicker**: This control allows users to select a date and time.

# Threading

Threading in programming allows a process to be divided into smaller parts, known as threads, that can run simultaneously. The advantages of threading include:

* **Responsiveness:** Allows a program to continue running even if part of it is blocked.
* **Resource Sharing:** Threads share the memory and resources of the process they belong to.
* **Economy:** It’s more economical to create and context switch threads.
* **Scalability:** Benefits increase in multiprocessor architecture.
* **Better Communication System:** Improved inter-process communication through thread synchronization.
* **Microprocessor Architecture Utilization:** Threads can execute in parallel on distinct processors.
* **Faster Context Switch:** Lower context switch time between threads compared to processes.

# Microservice

* is an architectural approach for building software applications as a collection of small, independent services that communicate through APIs, enabling flexibility and scalability.

# How comunicate between 2 services in microservice

To communicate between two services in a microservices architecture, you can use methods like HTTP/HTTPS RESTful APIs, message queues, gRPC, GraphQL, service meshes, direct database access, or WebSockets, depending on your specific needs and requirements.

# Key characteristics and principles of microservices include

**Decomposition:** Applications are divided into smaller, manageable services that can be developed by small teams. Each service should have a single responsibility or purpose.

**Independence:** Microservices are loosely coupled, meaning they can be developed, deployed, and maintained independently of each other. This allows for flexibility in technology stack, development cycles, and scalability.

**APIs:** Services communicate with each other through well-defined APIs (Application Programming Interfaces), typically over HTTP/REST or messaging protocols like Kafka or RabbitMQ.

**Scalability:** Services can be scaled independently based on their specific resource demands, improving resource utilization and application performance.

**Resilience:** Since microservices are isolated, failures in one service do not necessarily impact others. This can lead to more resilient and fault-tolerant applications.

**Continuous Delivery:** Microservices are often associated with DevOps practices, enabling rapid and continuous development, testing, and deployment.

**Technology Diversity:** Different services can use different technologies and programming languages that are best suited for their specific tasks.

**Data Management:** Handling data in a microservices architecture can be complex, with various approaches like distributed databases, event sourcing, and CQRS (Command Query Responsibility Segregation) used to manage data effectively.

**Monitoring and Observability:** Due to the distributed nature of microservices, monitoring and logging are crucial for understanding the health and performance of the entire system.

**Deployment and Orchestration:** Tools like containerization (e.g., Docker) and orchestration platforms (e.g., Kubernetes) are often used to simplify the deployment and management of microservices.

# How to hidden username and password of database in connectionString in .Net while deploy

Use Environment Variables

Save by Azure Key Vault or Similar Services

# Difference between code first and database first

Code First starts with defining entity classes in code and generates the database schema, while Database First starts with an existing database schema and generates entity classes.

# Entity Framework

* Entity Framework (EF) is a popular Object-Relational Mapping (ORM) tool that allows developers to interact with databases using an object-oriented programming approach

# Differences between Method Syntax and Query Syntax in Entity Framework

* **Method Syntax (Lambda Syntax):** Modern and concise, offers better IntelliSense support, and can be more efficient for complex queries. However, it might be harder to learn for newcomers.
* **Query Syntax:** Similar to SQL, making it easier for SQL developers to use. It’s often more readable for simple queries but can struggle with very complex ones.

Both are functionally identical and the choice often comes down to personal preference and specific project requirements.

# Difference between JavaScript and typescript

JavaScript is a dynamically typed language, while Typescript is a statically typed superset of JavaScript. Typescript adds static type checking, type annotations, and better tooling for improved code quality and maintainability.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Angular with JavaScript** | **Angular with TypeScript (Recommended)** |
| **Language** | JavaScript | TypeScript (superset of JavaScript) |
| **Static Typing** | No | Yes |
| **Errors** | May be caught at runtime | Often caught during development |
| **Code Clarity** | Lower due to lack of type information | Higher due to type annotations |
| **Maintainability** | Can be challenging in large projects | Easier due to early error detection |
| **Setup** | Simpler | Requires additional configuration |

# Logging in Web API

* Nlog, log4net, Serilog or Logging Extension of Microsoft

# Agile development methodologies keywords

**User Story:** A brief description of a product feature, told from the perspective of an end-user. User stories help define the functionality that should be developed.

**Product Owner:** The person responsible for prioritizing and managing the product backlog, representing the interests of the stakeholders, and ensuring that the team delivers value to the customer.

**Scrum Master:** The Scrum Master is a servant-leader who helps the Scrum team understand and adopt Agile practices and removes any impediments or obstacles that may hinder their progress.

**Sprint:** A time-boxed iteration during which a set of user stories or backlog items is developed, tested, and delivered. Typically, sprints last 2-4 weeks.

**Product Backlog:** A prioritized list of all the features, enhancements, and bug fixes that need to be developed in the product. It's maintained by the Product Owner.

**Velocity:** A measure of the amount of work a team can complete in a single sprint. It helps in planning and estimating future sprints.

**Daily Standup (Scrum):** A short, daily meeting where team members share progress, discuss obstacles, and plan for the day's work.

**Retrospective:** A meeting held at the end of each sprint to reflect on the team's performance and identify areas for improvement.

**Incremental Development:** Building a product in small, incremental steps, with each step delivering value and being potentially shippable.

**Customer Feedback:** Frequent interaction with customers or end-users to gather feedback and adjust the product accordingly.

**Continuous Integration (CI):** The practice of regularly integrating code changes into a shared repository, followed by automated testing, to ensure that the codebase is always in a functional state.

**Continuous Delivery (CD):** The practice of automatically deploying code changes to production or a staging environment after passing automated tests.

**Agile Manifesto:** A set of guiding values and principles for Agile software development, emphasizing customer collaboration, responding to change, and delivering working software.

**Kanban:** A visual management tool that helps teams visualize work, limit work in progress, and focus on completing tasks.

**Lean:** A philosophy and set of principles that focus on eliminating waste and optimizing processes to deliver value efficiently.

# Setting environment test in .NET core:

* .NET Core use an environment variable called ASPNETCORE\_ENVIRONMENT to indicate the run time environment
* The value of this variable can be anything as per you need but typically it can be develop, staging or production.
* We can set in properties, debug tab has field Environment variables or can change in launch setting JSON file.

# Controller routing:

* .NET Core controllers use the Routing middleware to match URLs of incoming request and map them to action
* Route template:
  + Are defined at startup in Program.cs
  + Describe how URL paths are matched to actions.
  + Are used to generate URLs for links. The generated link are typically returned in response.

# State management and Session management:

* State management:
  + Can be stored using several approaches.
  + Storage approach include :
    - Cookies
    - Session state
    - Temp data
    - Query strings
    - Hidden fields
    - HttpContext.Items
    - Cache
* Session state:
  + Is an ASP.NET Core scenario for storage of user data while the user browses a web app
  + Use a store maintained by the ap to persit data across request from a client.

# Top AWS services:

* EC2 (Elastic Compute Cloud)
* RDS (Relational Database Services)
* S3 (Simple Storage Service)
* Lambda
* SNS (Simple Notification Service)
* Etc

# Deploy to azure:

Publish to azure

# Azure Services Overview ([Azure Services Overview (azurecharts.com)](https://azurecharts.com/overview)):

* AI + Machine Learning
* Analytics
* Compute
* Databases
* Development
* Identity + Security
* IoT + MR
* Integration

# Authorization and authentication:

* Authentication:
  + Verifies who the user is
  + Works through password and other information user provided or entered by the user
  + Is the first step of a good identity and access management process
  + Is visible and partially changeable by the user
* Authorization:
  + Determine what resources a user can access
  + Works through setting are implemented and maintained by the organization
  + Always takes place after authentication
  + Invisible and not change by the user

# How to plan about the API

* We should discuss with Front End team and TA team to analysis requirement
* After that, we should create mockup test which we will base to handle API.

# Why use store Procedure?

* **Better performance** 
  + Calls are quick and efficient as stored procedures are compiled once and stored in executable form.
  + Hence response is quick
  + The executable code is automatically cached, hence lowers the memory requirements
* **Higher productivity**
  + Since the same piece of code is used again so, it results in higher productivity
* **Ease of Use**
  + Can be deployed on any tier of network architecture.
* **Scalability**
  + Increase scalability by isolating application processing on the server.
* **Maintainability**
  + Maintaining a procedure on a server is much easier then maintaining copies on various client machines, this is because scripts are in one location
* **Security**
  + Access to the Oracle data can be restricted by allowing users to manipulate the data only through stored procedures that execute with their definer’s privileges.

# Improve performance

* Use keyword Analyze to analyze an execution plan for a query, so that we can see query is slowly at what step and we can modify query to improve performance
* Restrict joins between tables
* Restrict relationships between tables

# You used token before? Authorization based on roles? What is best way to implement it?

* Yes, I have used token before
* I use **Identity Server 4** to generate token and use token to call API that is authorize.
* **Identity Server 4** library that support us custom authorize and authentication

# Have you hear clean architecture?

* I just read document about it, but I have never used it before

# Why we need use versioning

* Web API Versioning is required as the business grows and business requirement changes with the time.
* As Web API can be consumed by multiple clients at a time, Versioning of Web API will be necessarily required so that Business changes in the API will not impact the client that are using/consuming the existing API.

# Component in Angular

* Component in angular contain:
  + HTML file that declare what render on the page
  + A Typescript file that define behavior
  + A CSS selector that define how to component is used in a template
* Why use
  + Component reusability
  + Easy to do the error handling

# Can you explain pipe

* Pipe can transform the input return to the directly to the template and help you optimized the performance of the front end because it only fire when angular detect change

# Have you work with services?

* Service is a broad category encompassing any value, function, or feature that an application needs. A service is typically a class with a narrow, well-defined purpose

# Magic tables

Magic tables are the temporary logical tables that are created by the SQL server whenever there are insertion or deletion or update (D.M.L) operations and help improve the performance.

# Abstract class vs interface

* **Abstract class:**
  + Is a way to achive the abstraction in C#
  + To declare abstract class, we use abstract keyword
  + An abstract class is never intended to be instantiated directly
  + This class must contain at least one abstract method, which is marked by the keyword or modifier abstract in the class definition
  + The abstract classes are typically used to define a base class in the hierarchy.
* **Interface:**
  + Can have methods, properties, events, and indexers as it’s member
  + But interfaces will contain only declaration of the members
  + The implementation of interface’s member will be given by the class who implements the interface implicitly or explicitly

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| Multiple inheritance is not achieved by abstract class. | Multiple inheritance is achieved by interface |
| It contain constructor. | It does not contain constructor. |
| It can contain static members. | It does not contain static members. |
| It can contain different types of access modifiers like public, private, protected etc. | It only contains public access modifier because everything in the interface is public. |
| A class can only use one abstract class. | A class can use multiple interface. |

# Overloading and overriding

* Overloading:
  + means creating multiple method in a class with same names but different signatures
  + It permit a class, struct, or interface to declare mutilple methods with the same name with unique signatures
* Overriding:
  + Means having two methods with same name and same signature (parameter), one should be in the base class and other method should be in a derive class (child class)
  + Can override the functionally of a base class method to create same name method with the same signature in derived class
  + Achive method overriding using inheritance
  + Virtual and override keywords are used to achieve method overriding.

# Generics in C#

Generics in C# are a feature that allows you to define type-safe data structures, without committing to actual data types. This results in a significant performance boost and higher quality code, because you get to reuse data processing algorithms without duplicating code.

It’s using via T data type

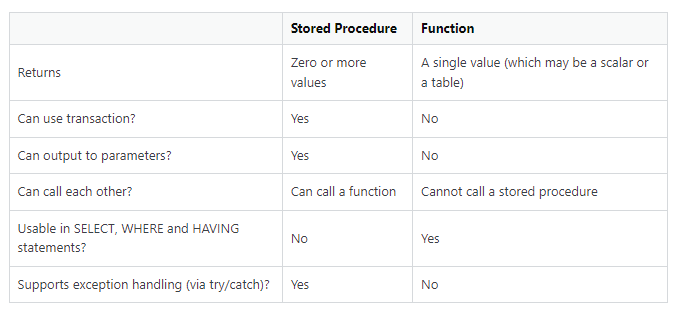
# Function in SQL server

* Are the database objects that contains a set of SQL statements to perform a specific task
* Accepts input parameters, perform actions, and then return the result.

# Stored Procedure

* Is a group of one or more pre-compiled SQL statements into a logical unit
* It is stored as an object inside the database server
* It is a subroutine or a subprogram in the common computing language that has been created and stored in the database
* Each procedure in SQL Server always contains a name, parameter lists, and Transact-SQL statements

# Different stored procedure and function



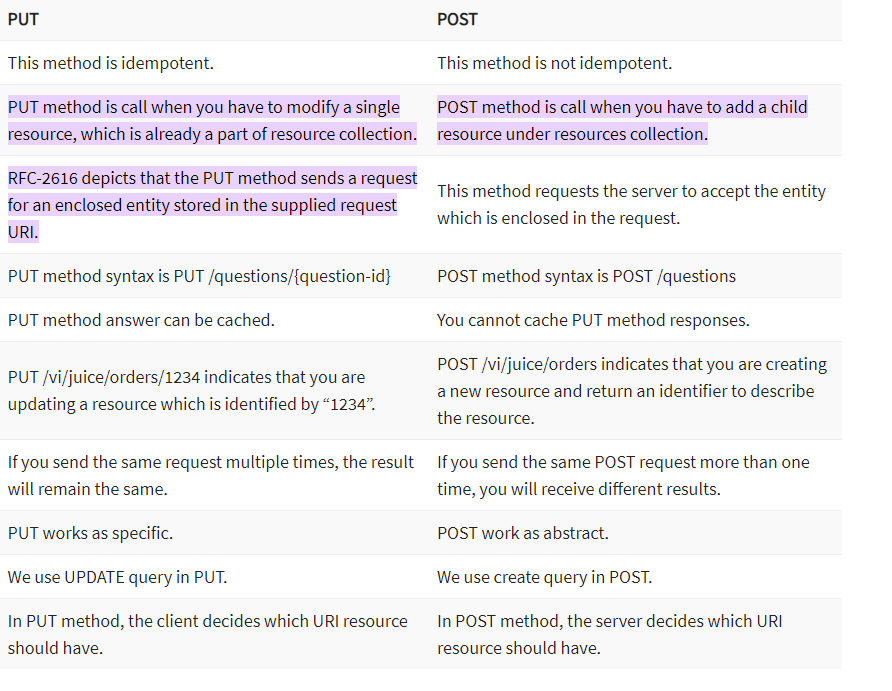
# Trigger

* Is a database object similar to a stored procedure that is executed automatically when an event occurs in a database
* There are different kinds of events that can activate a trigger like inserting or deleting rows in a table, a user logging into a database server instance, an update to a table column, a table is created, altered, or dropped, etc.

# Index in SQL

* Is a set of key made up of single or multiple column in the table or view
* They are stored in a structure that help SQL user quickly and efficiently find the row or rows associate with the key value

# PUT AND POST



# Singleton design pattern

* Define:
  + Is one of the most popular design pattern
  + In this pattern, a class has only one instance in the program that provide a global point of access to it
  + In other word, a singleton is a class that allow only single instance of itself to be created and usually give simple access to that instance
* Advantages:
  + Singleton pattern can implement interfaces
  + Can be lazy load and has static initialization
  + It help to hide dependencies
  + It provide a single point of access to particular instance, so it is easy to maintain
* Implement:
  + Usually use not thread safe singleton. In this implement, two different thread could both have evaluated the test if instance euqal null and found it to be true, then both create instance

# Repository pattern

The repository pattern is used to abstract data access logic from the rest of your application. It acts as an intermediary layer between your business logic (entities) and the data access layer (databases, files, etc.)

# Directive

* Are defined as classes that can add new behavior to the elements in the template or modify exsiting behavior

# Angular Module

* A module is a mechanism to group components, directives, pipes and services that are related in such a way that can be combined with other modules to create an application.

# NgModule

* NgModules configure the injector and the compiler and help organize related things together.
* The NgModule decorator requires at least three properties: imports, declarations and bootstrap.

# Extension method in C#

* Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type

# Middleware

* Middleware can perform work before and after the next component in the pipeline.

# Lazy loading

* The concept of lazy loading assists in loading only the required section and delays the remaining, until it is needed by the user.

# Annotation in Web API

* To set validation rules for properties on model

# How to convert JSON to object and vi versa

* using Newtonsoft.Json library (back-end)
* using JSON.PARSE (angular)

# Mình có entity với 10 attribute, làm sao chỉ expose 5 attribute (columns) to UI

* Use class DTO or Auto-mapper

# Configuration Service

* Add services to the container.
* In this method you can add config such as: cors, iis intergration, controller, swagger

# Setting index

* Data Annotations: use the Index keyword
* Fluent API: define in the method OnModelCreating with builder entity has index

# How you host your API?

* IIS, Cloud, …

# Which data type return from API

* JSON

# How we make sure we only return json format

* JsonResult

# How we handle exception in API

* Register Exception Filters and it will apply for all api.

# Have you deploy your code to IIS or not

* Sometime I deploy to IIS
* Normally, DevOps Team will do it

# How you secure your web API

* Authentication and authorization

# Token base authorization

* Is a protocol which allows users to verify their identity, and in return receive a unique access token

# How we handle authorization

* Authorize annotation

# How we implement if we want specific user with specific API

* Authorize annotation base on role of permission

# Static Class

* The static keyword is used to make a data item non-instantiable.
* It can be used with classes, methods, variables, constructors, operators etc.
* However, it cannot be used with destructors, indexers etc.

# Sealed Class

* In c#, sealed is a keyword used to stop inheriting the particular class from other classes.
* We can also prevent overriding the particular properties or methods based on our requirements.

# Partial Class

* The partial keyword indicates that other parts of the class, struct, or interface can be defined in the namespace.
* All the parts must use the partial keyword.
* All the parts must be available at compile time to form the final type.
* All the parts must have the same accessibility, such as public, private, and so on.

# What is CORS in web API

* Allows you to get away from the same-origin policy adopted by the browsers to restrict access from one domain to resources belonging to another domain
* You can enable CORS for your Web API using the respective Web API package (depending on the version of Web API in use).

# Web API testing tool

* Swagger, Postman

# Tool to troubleshoot performance issue in Web API

* WebLOAD, LoadNinja, HeadSpin

# API controller

* Web API Controller is similar to ASP.NET MVC controller. It handles incoming HTTP requests and send response back to the caller.
* The name of a controller class must end with "Controller" and it must be derived from System.Web.Http.ApiController class.

# Parameter binding

* Web API controllers can have one or more parameters of different types
* It can be either primitive type or complex type.
* Web API binds action method parameters with the URL's query string or with the request body depending on the parameter type.

# In request body, how we read data

* Model Mapping

# How many parts in JSON token

* Header. Payload. Signature
  + Header: type of token and the sign in algorithm being used such as: SHA256, RSA
  + Payload: contain claims. There are three types of claims: registered, public, and private claims.
  + Signature: To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

# How you authorize API method

* Authorize annotation

# HTTP response method

# Asp.net which framework

# How maintain logger

* can store in third-party lib
* elastic-search + Kibana

# How many log

# When you write log

* When runtime throw exception or the system throw exception

# Trace API throw exception

* Register exception filter and add it to the controller method to tracing the api when that throw exception

# When access huge data from DB

I can’t keep all data in session

I want to access data from different page.

The best way to handle

* Use cache
* Explain: improve the performance when query, instead of query from the database, we will query from the cache

# Decorators in Angular

Decorators are special functions that modify the behavior of classes and their parts. They essentially add functionalities or metadata to things like components, directives, services, and more.

# Filters in Angular

Filters are functions that specifically transform or manipulate data displayed in your templates. They help you narrow down information, format it differently, or customize how it appears.

# Directives in Angular

Directives are a fundamental concept for extending HTML functionalities. They act like instructions that manipulate the DOM (Document Object Model) or add new behaviors to existing HTML elements.

# Which data storage in cache

* Data with

# Why we use interface

* Interfaces are useful in object-oriented programming because they specify the behavior that an object must implement.
* An interface is a contract between an object and its code.
* If an object implements an interface, it is guaranteed to support the behavior specified by that interface.

# Why we use dependency injection

* The intent of Dependency Injection is to make code maintainable.
* Dependency Injection helps to reduce the tight coupling among software components. Dependency Injection reduces the hard-coded dependencies among your classes by injecting those dependencies at run time instead of design time technically.

# basic steps to implement dependency injecttion

# Unittest library

* UnitTesting + FluentAssertions

# JWT

* Is an open standard used to share security information between two parties – a client and server
* Each JWT contains encoded JSON object, including a set of claim
* JWT are signed using a cryptographic algorithm to ensure that the claims cannot be altered after the token is issued

# User Control

* I don’t know exactly user control in angular map to what, but I think it is a input and output that share data between child and parent directive and components

# Connect database

* Use entity framework and in the appsetting.json file add ConnectString with the name database that you want to connect.

# Restful API

* Is an architectural style for an application program interface (API) that uses HTTP requests to access and use this data
* That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources.

# Call store procedure from Entity Framework

* Through the database context use keyword SQL query to call the store procedure

# Pass data between component

* Use the input and output to share data between component
* Via the input can pass data from parent to child and output can pass data from child back to the parent

# Subcribe in the angular

* Connect the observer to observable event. Whenever any change is made in these observable, a code is executed and observes the results or change

# NgOnInit and Constructor

* The constructor should only be used to initialize class members but shouldn't do actual "work". So we should use constructor to setup Dependency Injection, Initialization of class fields etc.
* ngOnInit is a better place to write "actual work code" that we need to execute as soon as the class is instantiated.

# Setting environment in angular

* Use different config file

# Async pipe

* Display data async chornus
* Normally, use to load data from the server

# Configure method and configure service:

* ConfigureServices is used to configure Dependency Injection
* Configure is used to set up middlewares, routing rules, etc