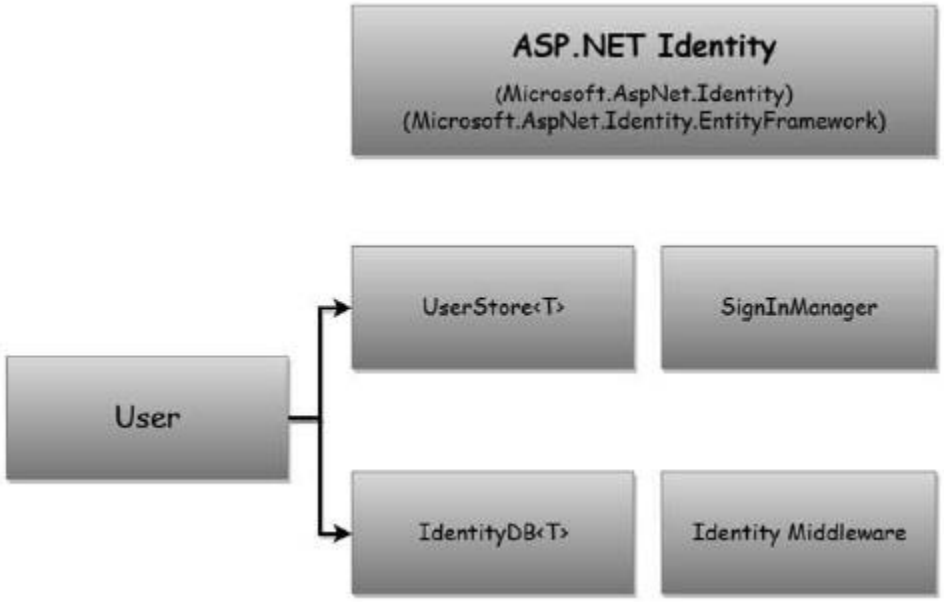
1. What is Asp.net core Identity? How to Add authentication to apps and identity service configurations?

ASP.NET Core Identity:

- Is an API that supports user interface (UI) login functionality.
- Manages users, passwords, profile data, roles, claims, tokens, email confirmation, and more.

Users can create an account with the login information stored in Identity or they can use an external login provider. Supported external login providers include Facebook, Google, Microsoft Account, and Twitter.

Identity is typically configured using a SQL Server database to store user names, passwords, and profile data. Alternatively, another persistent store can be used, for example, Azure Table Storage.



Steps to Add authentication to apps and identity service configurations:

Step 1: Install All Package from Nuget

EntityFrameworkCore

EntityFrameworkCore.SqlServer

EntityFrameworkCore.Tool

EntityFrameworkCore.Design

Step 2: Adding **AppDbContext** Inside Model Folder and inherits from **IdentityDBContext** Class

```
public class AppDbContext:IdentityDbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options):base(options)
        {

        }
        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

        }
    }
```

Step 3: Add Line Of code inside **Startup.cs** inside **ConfigureServaices Method**

```
services.AddDbContextPool<AppDbContext>(options => options.UseSqlServer("Database
connection string"));
services.AddIdentity<IdentityUser,IdentityRole>().AddEntityFrameworkStores<AppDbContext>(
);
```

Step 4: Add below line code inside **Startup.cs** inside **Configure** Method

```
app.UseAuthentication();
```

Step 5: Now, Go To Tools>Nuget Package Manager>Package Manager Console

Type: ***Add-Migration AddingIdentity***

Step 6: Type**: Update Database**

2.Short Notes

Cross Site Scripting (XSS) in ASP .NET Core:

Cross Site Scripting (XSS) is an attack where attackers inject code into a website which is then executed. Cross site scripting is the injection of malicious code in a web application, usually, Javascript but could also be CSS or HTML. When attackers manage to inject code into your web application, this code often gets also saved in a database. This means every user could be affected by this. For example, if an attacker manages to inject Javascript into the product name on Amazon. Every user who opens the infected product would load the malicious code.

There are many possible consequences for your users if your website got attacked by cross site scripting:

1. Attackers could read your cookies and therefore gain access to your private accounts like social media or bank

2. Users may be redirected to malicious sites
3. Attackers could modify the layout of the website to lure users into unintentional actions
4. Users could be annoyed which will lead to damage to your reputation and probably a loss of revenue
5. Often used in combination with other attacks like cross site request forgery (CSRF)

Preventing XSS attacks is pretty simple if you follow these best practices:

1. Validate every user input, either reject or sanitize unknown character, for example, < or > which can be used to create
2. Test every input from an external source
3. Use HttpOnly for cookies so it is not readable by Javascript (therefore an attacker can't use Javascript to read your cookies)

Use markdown instead of HTML editors

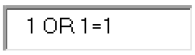
SQL INJECTION:

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.
- SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: 

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 1 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

The SQL statement above is much the same as this:

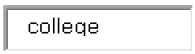
```
SELECT UserId, Name, Password FROM Users WHERE UserId = 1 or 1=1;
```

SQL Injection Based on ""="" is Always True

Here is an example of a user login on a web site:

Username:

Password:



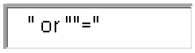

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' +
uPass + '"'
```

Result

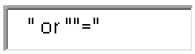
```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:

User Name:



Password:



The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since **OR ""=""** is always TRUE.

Use SQL Parameters for Protection

To protect a web site from SQL injection, you can use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

ASP.NET Razor Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = @0";
db.Execute(txtSQL,txtUserId);
```

Note that parameters are represented in the SQL statement by a @ marker.

The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

Another Example

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City)
Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

CSRF In ASP.NET Core

CSRF or Cross Site Request Forgery is a type of web attack that uses a users own browser to post a form from one site to another. It works like so :

- User logs into www.mybankaccount.com and receives a cookie.
- Sometime later the user goes to www.malicioussite.com and is shown a completely innocuous form on the surface. Something like :

4.What is open redirect attack how to prevent it?

An Open Redirection is when a web application or server uses a user-submitted link to redirect the user to a given website or page.

How can I identify is my application is vulnerable or not?

1. If your application redirects to URL which is directly given by user that's specified via the request such as query string or form data.
2. The redirection is performed without checking if the URL is a local URL.

```
public async Task<IActionResult> Login(LoginViewModel model,string

returnUrl)

{

}
```

n this code we pass this URL directly to the Redirect. We never check if the URL is local or not, meaning that our application is vulnerable to open redirect attacks.

https://ourwebsite.com/account/login?returnURL=http://hackerwebsite.com/account/login

See in the above URL where the first part is our website and in return, the URL is given by a hacker which could be malicious or steal data. If you see the first part it looks like your website and generally, we wouldn't look at the second part which the hacker could use to easily redirect us to their site.

To Prevent Open Redirect Attacks

LocalRedirect In Asp.Net Core

Rather than using Redirect, use LocalRedirect so when the user tries to add another domain URL it will prevent it and give an error.

See in the above image that we used Local redirect in our code. When we login, I pass the return URL as =https://google.com which is not local and our complete URL as below,

https://localhost:44387/Account/Login?ReturnUrl=https://google.com

So it will throw an error like below:

Exception Message

The supplied URL is not local. A URL with an absolute path is considered local if it does not have a host/authority part. URLs using virtual paths (‘~/’) are also local.

As we *handle error globally* so that’s why such page and message occurs.

Url.IsLocalUrl In Asp.Net Core

If you want to use Redirects only then you can check the URL first and then perform a redirection. The code for checking the URL is shown below

```
Url.IsLocalUrl(returnUrl)
```