

# FIT5032

# Internet Applications Development

Week 7: Security and Identity  
Murray Mount / ABM



[www.shutterstock.com](http://www.shutterstock.com) · 621799472

# Unit Topics

Week	Activities	Assessment
0		No formal assessment or activities are undertaken in week 0
1A	Intro to Web development and ASP.NET	Note: Studio classes commence in week 1
1B	The front end, user experience, accessibility and ASP.NET Scaffolding	
2	Introduction to C# & Version Control	
3	Entity Framework	
4	Fundamentals of Client side Javascript	Studio assessment task 1 due
5A	Validation	
5B	Security and Identity	
6	Sending Email, File Upload and Signal R	Studio assessment task 2 due
7	Web Optimisations & Evolution of ASP.NET CORE	
8A	Modern JavaScript Web Development Approaches	
8B	Testing and Deployment in Cloud	Studio assessment task 3 due
9	Review & Revision	Final Portfolio and Learning Summary due
	SWOT VAC	No formal assessment is undertaken in SWOT VAC
	Examination period	<a href="http://policy.monash.edu.au/policy-bank/academic/education/assessment/assessment-in-coursework/">LINK to Assessment Policy: http://policy.monash.edu.au/policy-bank/academic/education/assessment/assessment-in-coursework/</a>

# Today

- Recap: Validation in ASP.NET MVC
- Security and Identity
- Log In Features
- Security Issues (in example)
- Additional Security Topics: XSS, CSRF, HTTP Headers
- OpenID, OAuth

# Recap: Validation

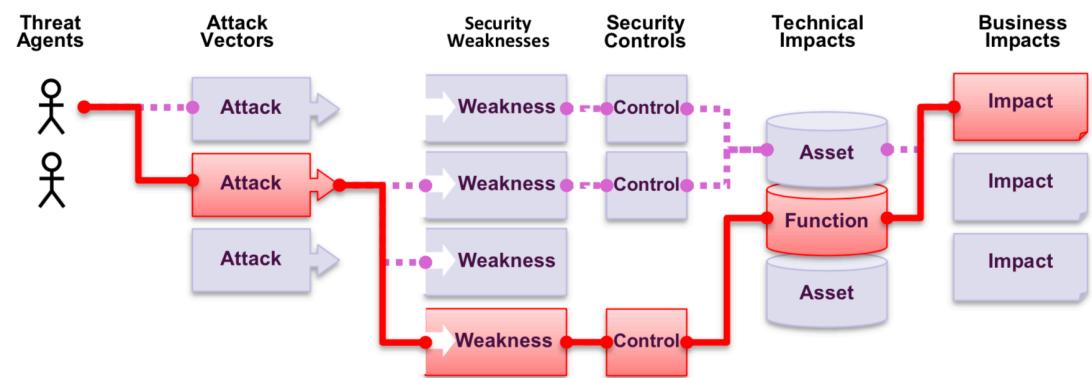
# Validation in ASP.NET MVC

- Main aspects to implementing validation in ASP.Net MVC Applications
  - Validation in Models
  - Validation in Views
  - Validation Error Messages
  -

# Security and Identity

# OWASP Top 10 Web Application Security Risks

1. Injection
2. Broken Authentication and Session Management
3. Sensitive Data Exposure
4. XML External Entity
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting
8. Insecure deserialization
9. Using Components With Known Vulnerabilities
10. Insufficient Logging and Monitoring



<https://www.owasp.org/>

# Multi factor authentication

- The authentication factors that make up a multi-factor authentication request must come from two or more of the following:
  - something the claimant knows (e.g. a personal identification number (PIN), **password** or response to a challenge) **(Something you know)**
  - something the claimant has (e.g. a physical token, smart card or software certificate) **(Something you have)**
  - something the claimant is (e.g. a fingerprint or iris scan). **(Something you are)**

# Hashing

- Hashing is an ideal way to store **passwords**, as hashes are inherently one-way in their nature.
- When storing a password, **hash it with a salt**, and then with any future login attempts, hash the password the user enters and compare it with the stored hash.
- If the two match up, then it's virtually certain that the user entering the password entered the right one.
- MS Identity uses the **PBKDF2** to hash passwords.

# ASP.NET Identity Goals

- One ASP.NET Identity system
- Ease of plugging in profile data about the user
- Persistence control
- Unit testability
- Role provider
- Claims based
- Social Login Providers
- Azure Active Directory
- OWIN Integration
- NuGet package

# Log In Features

# Log In Concepts

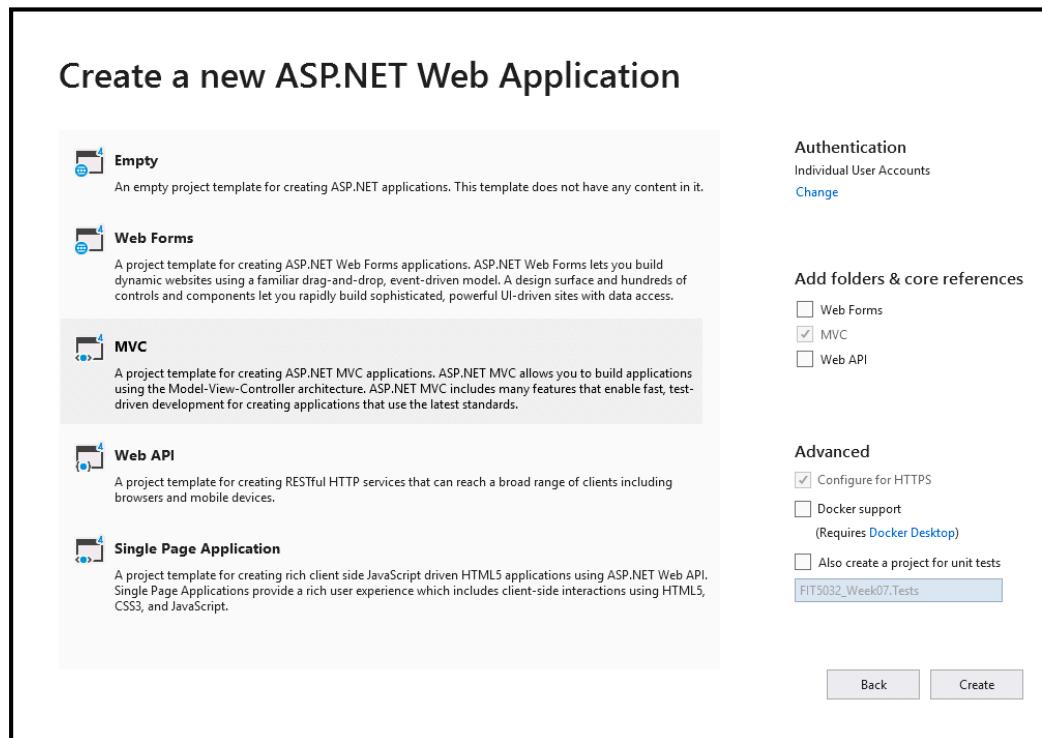
- Almost all real world web applications require users to log in to the website
  - to use more than the basic functionality.
- Require **usernames** and **passwords**
- Some applications use role based **authentication**
  - administrator roles, user roles etc
- Security and account information stored
  - on file system
  - or database

# Log In Systems for ASP.Net MVC

- ASP.Net MVC application
  - Can auto-generate applications with log in functionality
- Basic ASP.Net MVC application
  - with users
    - register
    - interact with public areas before log in
    - interact with private areas after log in

# Creating an application with log in accounts

- Auto-generate applications with log in functionality



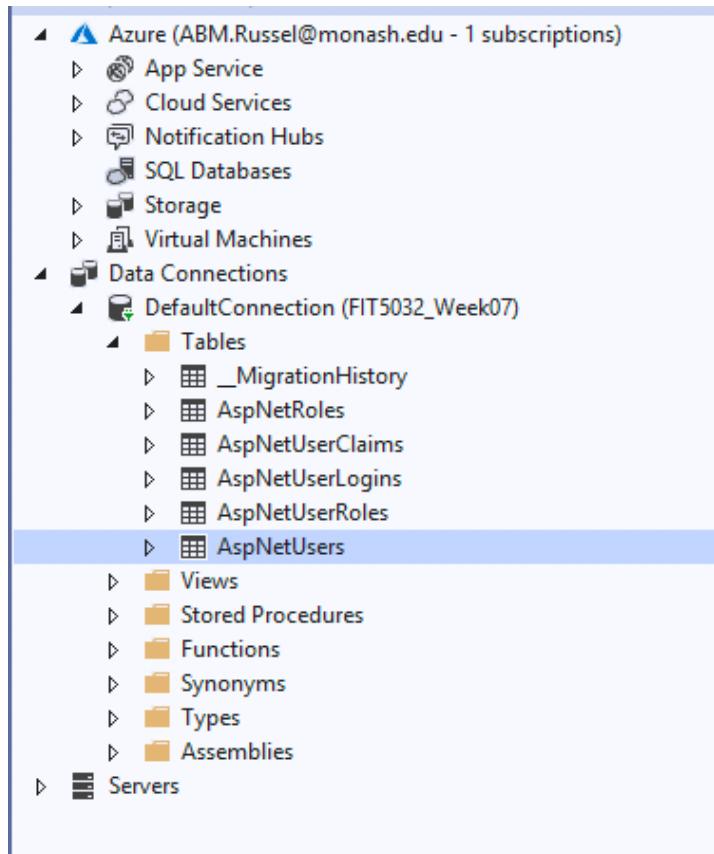
# Registering a user

- You can **register** a user in this new application that you created

The screenshot shows a registration form titled "Register." at the top left. Below the title, it says "Create a new account." There are three input fields: "Email" (with a placeholder), "Password" (with a placeholder), and "Confirm password" (with a placeholder). A "Register" button is located below the password fields. At the bottom left of the form, there is a copyright notice: "© 2019 - My ASP.NET Application". The top navigation bar includes links for "Application name", "Home", "About", "Contact", "Register" (which is highlighted in blue), and "Log in".

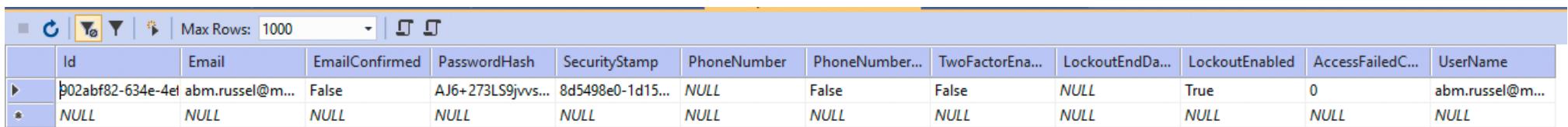
# User Account Tables

- Several Tables are created when the application is autogenerate



# User Table

- The **AspNetUser** table has the details of the user that have registered



A screenshot of a database management tool interface, likely SQL Server Management Studio (SSMS). The title bar shows the window is titled 'User Table'. The toolbar includes standard icons for file operations, refresh, and search. A dropdown menu 'Max Rows: 1000' is visible. The main area displays a table with the following columns: Id, Email, EmailConfirmed, PasswordHash, SecurityStamp, PhoneNumber, PhoneNumber..., TwoFactorEna..., LockoutEndDa..., LockoutEnabled, AccessFailedC..., and UserName. There are two rows of data. The first row is highlighted with a yellow background, indicating it is selected. The second row is standard white.

	Id	Email	EmailConfirmed	PasswordHash	SecurityStamp	PhoneNumber	PhoneNumber...	TwoFactorEna...	LockoutEndDa...	LockoutEnabled	AccessFailedC...	UserName
▶	b02abf82-634e-4ef abm.russel@m...	False	AJ6+273LS9jvvs...	8d5498e0-1d15...	NULL	False	False	NULL	True	0	abm.russel@m...	
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

# Log In the User

- The Registered Users can Log in

Log in.

[Register](#) [Log in](#)

Use a local account to log in.

Email

Password

Remember me?

[Log in](#)

[Register as a new user](#)

© 2019 - My ASP.NET Application

Hello abm.russel@monash.edu! [Log off](#)

# Securing an Action

- An Action (e.g. from the HomeController) can be restricted to logged in users
  - Use the [Authorize] annotation

[Authorize]

```
public ActionResult Contact()  
{  
    ViewBag.Message = "Your contact page.";  
  
    return View();  
}
```

- Now the user must log in to access the Contact action.

# Roles

- Normally applications are secured based on roles
  - Administrators
  - Authors
  - general public
  - Etc...

# ASP.Net MVC Roles

- We can specify roles in the application by adding entries in the **AspNetRoles** table

The screenshot shows the Server Explorer and Object Explorer windows in Visual Studio.

**Server Explorer:** Shows Azure subscriptions, Data Connections (DefaultConnection), and various database objects like Tables, Views, and Procedures. The **AspNetRoles** table is selected in the Object Explorer.

**Object Explorer:** Shows the structure of the **DefaultConnection** database, including the **AspNetRoles** table. The table data is as follows:

	Id	Name
	1	Administrator
	2	Author
▶*	NULL	NULL

# Mapping Users and Roles

- The Mapping of the Users to the roles is done in the **AspNetUserRoles** table

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Server Explorer pane displays a tree view of Azure resources, Data Connections, and the DefaultConnection (FIT5032\_Week07) database, with the AspNetUserRoles table selected. On the right, the Object Explorer pane shows the structure of the AspNetUserRoles table, which has two columns: UserId and RoleId. The data grid displays one row where the UserId is a GUID and the RoleId is 1.

	UserId	RoleId
*	902abf82-634e-4ef6-bcd2-7b01c4...	1
	NULL	NULL

# Intro to ASP.Net security

- Sections Secured by restricting controllers
  - Add the "[Authorize]" annotation in the HomeController.cs

```
namespace FIT5032_Week07.Controllers
{
    [Authorize]
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        [Authorize]
        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
}
```

# Securing/Unsecuring Actions

- Smaller sections secured
  - adding the “[Authorize]” annotation to the action.
- Secured controller, can have unsecured action
  - “[AllowAnonymous]”annotation for that action.

# Securing Controllers/Actions based on roles

- Application (controllers and actions)
  - secured using the roles
  - defined for the application (in the AspNetRoles table)
- Use “[Authorize(Roles = “Administrator”)]”
  - name of the roles are your choice.

# Multiple Roles

- Multiple roles allowed access
  - `[Authorize(Roles = "HRManager,Finance")]`
- Multiple roles needed to access
  - `[Authorize(Roles = "PowerUser")]`
  - `[Authorize(Roles = "ControlPanelUser")]`
- Reference: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles>

# Allowing Access to Own Data (Only)

- Selecting/Viewing items owned by log in user
  - ASP.Net MVC allows us to access the currently logged in user:

```
using Microsoft.AspNet.Identity;
```

```
.....
```

```
string currentUserId = User.Identity.GetUserId();
```

```
.....
```

# Selecting/Viewing items owned by log in user (Part 2)

- User id to select just the items that are created by the user (for viewing in the index view.)

```
// GET: Articles
public ActionResult IndexUserNames()
{
    //return View(db.Articles.ToList());
    string currentUserId = User.Identity.GetUserId();
    return View(db.Articles.Where(m=> m.AuthorId ==
currentUserId).ToList());
}
```

# Selecting/Viewing items owned by log in user (Part 3)

- Only the users own data is shown

The screenshot shows a web application interface. At the top, there is a navigation bar with links: Application name, Home, About, Contact, Articles, and My Articles. On the right side of the navigation bar, it says "Hello abm.russel@monash.edu" and "Log off". Below the navigation bar, the page title is "Index". There are two "Create New" buttons: one for general use and one specifically for the logged-in individual. The main content area displays a table of articles. The table has two columns: "AuthorId" and "ArticleText". Each article row includes "Edit | Details | Delete" links. The data in the table is as follows:

AuthorId	ArticleText	
d0e94786-e882-47f1-940a-9c4da5a59408	Hello	Edit   Details   Delete
d0e94786-e882-47f1-940a-9c4da5a59408	Testing	Edit   Details   Delete
d0e94786-e882-47f1-940a-9c4da5a59408	How are you	Edit   Details   Delete
d0e94786-e882-47f1-940a-9c4da5a59408	This is a new article	Edit   Details   Delete
c69eaf4-58f1-4e55-81f0-a651268dbf77	FIT5032 is a fun course	Edit   Details   Delete
8405f3e9-02fa-4b4b-88cc-db760f3c2a32	Have a great day!	Edit   Details   Delete

# Creating item (automatically adding userID)

- Action takes the current user id and adds it to the model before calling the View (with the model)

```
// GET: Articles/Create
```

```
public ActionResult CreateIndividual()
{
    Article article = new Article();
    string currentUserId = User.Identity.GetUserId();
    article.AuthorId = currentUserId;
    return View(article);
}
```

# Creating item (automatically adding userID) Part 2

- Process the Posted Model Values when the user Completes the form

```
[HttpPost]  
[ValidateAntiForgeryToken]  
public ActionResult CreateIndividual([Bind(Include = "NewsId, AuthorId,  
ArticleText")] Article article)
```

.....

# Creating item (automatically adding userID) Part 3

Application name Home About Contact Articles My Articles Hello abm.russell@monash.edu Log off

## Index

Create New

Create New (logged in individual) 

AuthorID	ArticleText	
d0e94796-e682-47f1-940a-9c4da5a59400	Hello	Edit   Details   Delete
d0e94796-e682-47f1-940a-9c4da5a59400	Testing	Edit   Details   Delete
d0e94796-e682-47f1-940a-9c4da5a59400	How are you	Edit   Details   Delete
d0e94796-e682-47f1-940a-9c4da5a59400	This is a new article	Edit   Details   Delete
c69ear04-58f1-4e55-81f0-a651258d9f77	FIT5032 is a fun course	Edit   Details   Delete
8405f3e9-02fa-4b4b-88cc-db760f3c2a32	Have a great day!	Edit   Details   Delete

Application name Home About Contact Articles My Articles

## CreateIndividual

### Article

AuthorID 8405f3e9-02fa-4b4b-88cc-db760f3c2a32;

ArticleText

Create

[Back to List](#)

# Creating item (automatically adding userID) Part 4

- A hidden field (model.AuthorId) is required to pass the AuthorID to the user.

```
@Html.HiddenFor(model => model.AuthorId,  
htmlAttributes: new { @class = "form-control"})
```

- Normally the internal Id values (such as AuthorID) are not displayed in the user interface, this is just for debugging purposes.

The screenshot shows a web page titled "CreateIndividual Article". At the top, there is a navigation bar with links: "Application name", "Home", "About", "Contact", "Articles", and "My Articles". Below the navigation bar, the main content area has a heading "CreateIndividual Article". Underneath the heading, there is a form with two fields. The first field is labeled "AuthorId" and contains the value "5405f3e9-02fa-4b4b-88cc-db760f3c2ad2". The second field is labeled "ArticleText" and contains an empty text input box. Below the input box is a "Create" button. At the bottom of the form, there is a link "Back to List".

# Role and User based Security

- With these approaches:
  - Users can be restricted to their own data
  - Administrators given write access to all data
  - Public (no logged in users) given read access only

# What is role based authentication?

## **What is role based authentication?**

- A. Using Usernames and Passwords to log in
- B. Using roles such as administrator, user etc
- C. Mapping of usernames to roles
- D. All of the answers (except none)
- E. None of the answers

# What is the most secure way of storing passwords?

**What is the most secure way of storing passwords?**

- A. No hashing
- B. Hashing
- C. Salting
- D. Salting and Hashing
- E. All of the answers

In ASP.Net MVC, how do you require users to log in to the system?

**In ASP.Net MVC, how do you require users to log in to the system?**

- A. Annotate the action with [Authorize(Roles = "Administrator")]
- B. Annotate the action with [Authorize]
- C. Map the username to the role
- D. All of the answers (except none)
- E. None of the answers

In ASP.Net how do you restrict a section of the application to a specific role (e.g. Administrator)

**In ASP.Net how do you restrict a section of the application to a specific role (e.g. Administrator)**

- A. Annotate the action with [Authorize(Roles = "Administrator")]
- B. Annotate the action with [Authorize]
- C. Annotate the action with [AllowAnonymous]
- D. All of the answers (except none)
- E. None of the answers

In ASP.Net MVC, how do you restrict the logged in user to just see their own data?

**In ASP.Net MVC, how do you restrict the logged in user to just see their own data?**

- A. This is done automatically in ASP.Net MVC
- B. Filter the model with ".Where(m=> m.Id == currentUserId)"
- C. Filter the view to just show the relevant data
- D. All of the answers (except none)
- E. None of the answers

# ASP.NET MVC Security Issues

# Security Weakness #1

- The default CRUD actions have not been secured
  - When new functionality is added (new Actions) and secured, make sure the default CRUD actions are **secured** (for administrator access)
- Or remove the default actions

# Security Weakness # 2

- The index action lists only items that are related to the user, including edit, delete links

Index		
<a href="#">Create New</a>		
<a href="#">Create New (logged in individual)</a>		
AuthorId	ArticleText	
d0e94786-e882-47f1-940a-9c4da5a59408	Hello	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
d0e94786-e882-47f1-940a-9c4da5a59408	Testing	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
d0e94786-e882-47f1-940a-9c4da5a59408	How are you	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
d0e94786-e882-47f1-940a-9c4da5a59408	This is a new article	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
c69eaf4-58f1-4e55-81f0-a651268dbf77	FIT5032 is a fun course	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
8405f3e9-02fa-4b4b-88cc-db760f3c2a32	Have a great day!	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

- However any user can **directly access** the edit/delete action using the url !
  - They just need to know the item id

# Security Weakness # 2

- Edit/Delete actions need to check user id:
  - Ensure the item that is being edited/deleted has a matching user ID to the current user!

# Additional Security Tips

# Cross Site Scripting (XSS)

- ASP.Net MVC has built in checks to ensure scripts are not submitted via user input
- However if you want users to input scripts (say a programming site)
- Disable validation for the relevant action  
`[ValidateInput(false)]`

# Cross Site Request Forgery (CSRF) Attack

- It is possible for another site to submit (normally malicious) data to your application
  - ASP.Net MVC has a built in functionality to stop this
  - Add `@Html.AntiForgeryToken()` to the form
  - Add `[ValidateAntiForgeryToken]` to the action method

# Custom HTTP Headers

- **X-Frame-Options**
  - Don't allow iframe.
- **X-Xss-Protection**
  - Stop loading the page when a cross-site scripting attack is detected.
- **X-Content-Type-Options**
  - Harder for hackers to guess the right mime type of resources.

# Custom HTTP Headers

- [Referrer-Policy](#)
  - Removes the referrer from http request, so difficult for admin from sites your users click to see referrer details. Also minimises what you can analyse about traffic flow on your site.
- [Strict-Transport-Security](#)
  - Only HTTPS allowed once this is set correctly
- [X-Powered-By](#)
  - Hides the technology used for your site.

# Custom HTTP Headers

- X-AspNetMvc-Version
  - Hides MVC version.
  - `MvcHandler.DisableMvcResponseHeader = true;`
- Server
  - Hides the IIS server details.
  - `HttpContext.Current.Response.Headers.Remove("Server");`
- Additional settings in:
  - Content-Security-Policy

You have created a default ASP.Net MVC CRUD application with user accounts. You've created a new action ListMine (annotated

**You have created a default ASP.Net MVC CRUD application with user accounts. You've created a new action ListMine (annotated with the [Authorize] keyword) that shows all the items for the current user. Identify one potential security problem?**

- A. Anybody can use the ListMine
- B. The ListMine action should be annotated with [Authorize(Roles = "Administrator")]
- C. The index action still lists all values
- D. All of the answers (except none)
- E. None of the answers

You have created a default ASP.Net MVC CRUD application with user accounts. You've created a new action ListMine (annotated

**You have created a default ASP.Net MVC CRUD application with user accounts. You've created a new action ListMine (annotated with the [Authorize] keyword) that shows all the items for the current user. Identify a second potential security problem?**

- A. The delete action should be annotated with [Authorize(Roles = "Administrator")]
- B. The userID should be checked with the edit and delete actions
- C. The edit action should be annotated with [Authorize(Roles = "Administrator")]
- D. All of the answers (except none)
- E. None of the answers

# What is Cross Site Scripting Attack?

## What is Cross Site Scripting Attack?

- A. When scripts are submitted to your site (potentially executing)
- B. When hackers log into your machine and change the scripts
- C. When hackers get angry at your scripting methods
- D. All of the above (except none)
- E. None of the above

# What is Cross Site Request Forgery

## What is Cross Site Request Forgery

- A. When another site looks like your site
- B. When another site submits malicious input to your site
- C. When another site has code similar to your site
- D. All of the above (except none)
- E. None of the above

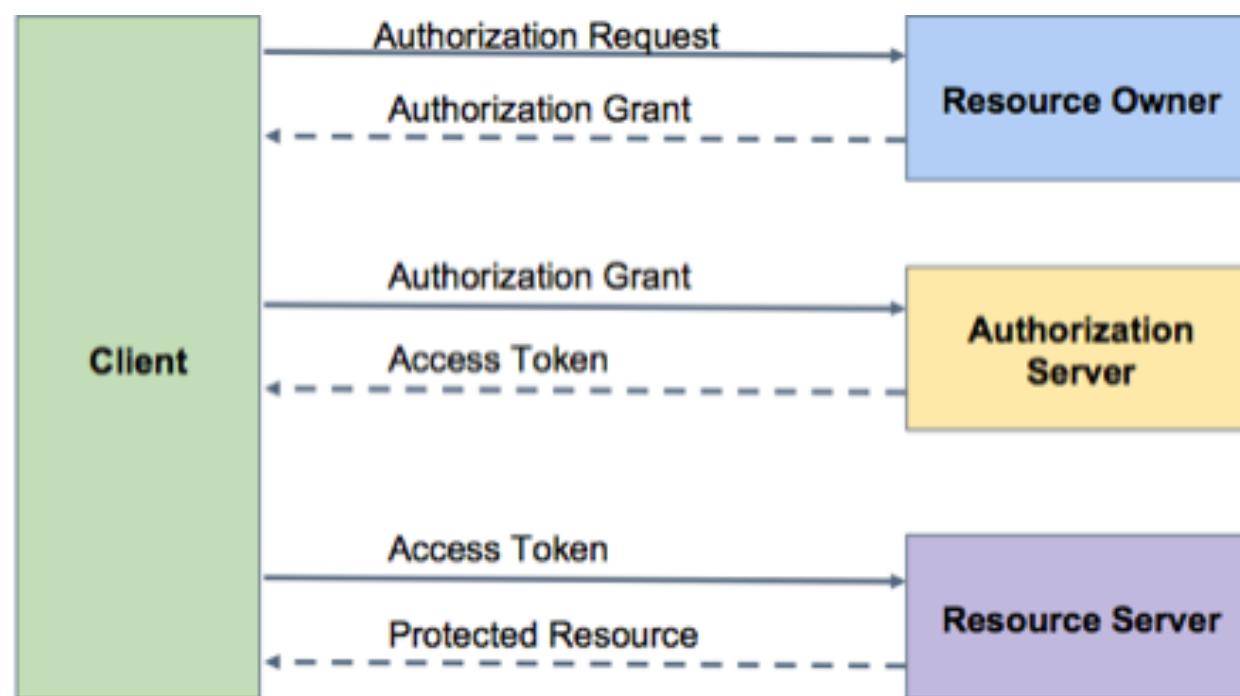
# OpenID, OAuth

# OpenId

- OpenID is an open standard and decentralized **authentication** protocol.
  - It allows Clients to **verify the identity** of the End-User based on the **authentication** performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.
- Remember one **username** and one **password**.

# OAuth

- OAuth 2.0 focuses on client developer simplicity while providing specific **authorization** flows for web applications, desktop applications, mobile phones, and living room devices.



# List of OAuth Providers

- Amazon
- BattleNet
- BitBucket
- Dropbox
- Github
- Google
- Instagram
- LinkedIn
- Twitter
- Yahoo



# Magic Quadrant for Access Management

Figure 1. Magic Quadrant for Access Management



Source: Gartner (August 2019)

# References

- <https://www.asp.net/mvc/overview/security>
- <http://www.csharpcorner.com/UploadFile/cda5ba/security-feature-in-mvc/>
- <https://blog.elmah.io/improving-security-in-asp-net-mvc-using-custom-headers/>
-

An action (from HomeController) can be restricted to logged users using which annotation

**An action (from HomeController) can be restricted to logged users using which annotation**

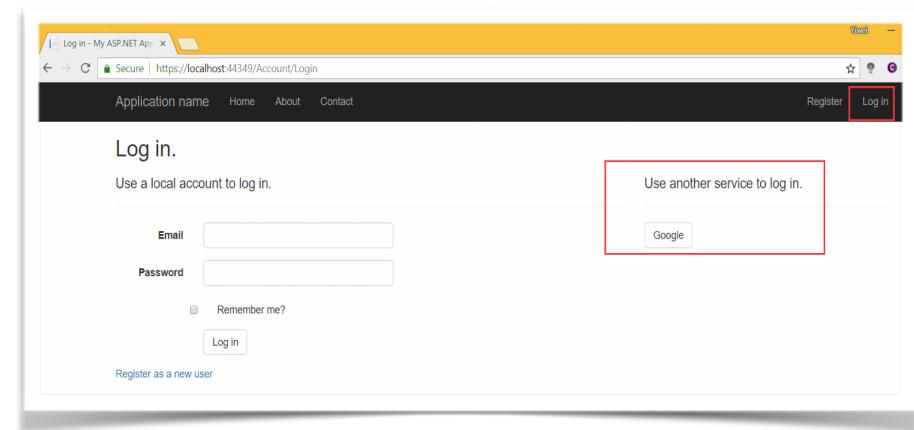
- A. [Authentication]
- B. [Login]
- C. [Authorize]
- D. [Security]
- E. [None]

# Lecture Summary

- Recap: Validation in ASP.NET MVC
- Security and Identity
- Log In Features
- Security Issues (in example)
- Additional Security Topics: XSS, CSRF, HTTP Headers
- OpenID, OAuth

# Week 7 Studio Overview

- Security Presentation
- Using ASP.NET Identity
- MS Identity to sign in with Google Account



# Next week: Email, File Upload, Signal R

- Sending Email
- File Upload
- Signal R