# Contents

# Chapter 1: ASP.NET MVC Basic

## Question 1

Suggest an appropriate software stack (operating system, web server, database server and programming runtime) for hosting ASP.NET web applications.

(lecture note pg 4)

Software stack: 4 core software components for hosting and running web applications:

- Operating System
- Web Server
- Database
- Programming Runtime

| | ASP.NET | PHP | Java |
|---|---|---|---|
| Operating System | Windows Server | Linux | Linux |
| Web Server | Internet Information Services (IIS) | Apache | Tomcat |
| Database | SQL Server | MySQL | Derby |
| Programming Runtime | ASP.NET (C#) | PHP | Java |

## Question 2

Discuss the advantages of using the ASP.NET platform for developing web applications from the following aspects:

- Existing software environment
- Customer support
- Expandability and scalability
- Rich and unified frameworks
- Consistent software components

- Existing software environment
    - Many companies may already using Windows-based software. (high familiarity)
    - It is easy, fast and more cost-effective to apply ASP.NET. (accessibility)
    - There is no need for buying and deploying new software stack as the existing software environment is sufficient to work.
    - No need to train system admins with new skill as they may already familiar with Windows-based systems. (low learning-time)

- Customer support
    - Windows-based software are usually paid or proprietary专利 so they have better customer support. (paid so better customer support)
    - Customer support is included in software license agreement.
    - No extras cost incurred if already purchased the software license.
    - Free software does not cover customer support (need extra cost to get customer support)
- Expandability and scalability
    - Many Microsoft products are designed to be enterprise-graded including software stack for ASP.NET.
    - Thus, it allows companies and businesses to grow, expand and upgrade to enterprise-level software infrastructure in future.
- Rich and unified frameworks
    - ASP.NET supports different web frameworks which can be integrated easily.
    - ASP.NET MVC provides web development using MVC architecture.
    - Each supports different sets of web development requirements.
- Consistent software components
    - Microsoft provides consistent software tools for the development and deployment of ASP.NET web applications.
    - Visual Studio IDE can be easily used for developing ASP.NET web applications.
    - Microsoft Windows Server acts as an enterprise-graded operating system.
    - Microsoft SQL Server acts as an enterprise-graded database server.

## Question 3

Identify and discuss the role of the 3 core components of MVC web architecture. Provide 2 examples for each.

| Model | View | Controller |
|---|---|---|
| Data layer | Presentation layer | Application logic layer |
| Represent data of the application | Template files for application uses | Handle incoming requests |
| Use validation rules to enforce business rules for the data | Generate HTML response dynamically | Retrieve model data, specify view template and return responses |
| e.g. Database, entity classes | e.g. HTML, CSS | e.g. C# controller classes, action methods |

## Question 4

Discuss the advantages of MVC web architecture from the following aspects:

- Separation of concerns

- Task distribution
- Maintainability
- Reusable
- Unit Testable

- Separation of concerns
    - Each MVC components focuses on specific tasks (modular)
    - Making each component simple and easy to program, debug and maintain
- Task distribution
    - Easy to distribute tasks among developers, database administrators and designers
    - Developers: focus on **controllers**
    - Database administrators: focus on **models**
    - Designers: focus on **views**
- Maintainability
    - Modification on MVC component usually has no or minimum effects to other components
    - Making it easy to maintain
- Reusable
    - MVC application is modular
    - Allows MVC components (especially model and view) to be reused in other project with minimum modifications
- Unit testable
    - Application logic (controller) is decoupled from data and presentation
    - Making it easy to employ automated unit testing to MVC application

## Question 5

Discuss how MVC web architecture help in task distribution for a project team that involves web developers, web designers and database administrators.

### Task distribution

- Easy to distribute tasks among developers, database administrators and designers
- Developers: focus on **controllers**
- Database administrators: focus on **models**
- Designers: focus on **views**

## Question 6

Discuss why URL routing is necessary for ASP.NET MVC web applications.

- Web pages are not physical files but it is the result of interaction between model, view and controller.
- URL routing engine maps incoming requests (URLs) against the routing rules.
- It can help for identifying the targeted **controller** and **action method** to be called.

## Question 7

Discuss in words, how the following URL is mapped to the respective components in ASP.NET MVC:

https://www.abc.com/Product/Update/P001?quantity=123

- Controller: Product
- Action: Update
- Parameters passed
    - Product id: P001
    - Quantity: 123

The URL maps to Product controller and Update action method by passing the product id and quantity parameters.

## Question 8

Given the domain is https://www.abc.com. Write the URL to access to the following controller and action method. Pass S001 for **id** and **2024** for **year**.

```
Public class StudentController : Controller
{
        public IActionResult Detail(string id, int year)
        {
                …
        }
}
```

https://www.abc.com/Student/Detail/S001?year=2024

### Question 9

Rewrite the following codes by filling in the missing part so that it maps to the following URL:

https://www.abc.com/Product/Update/P001?quantity=123

```
public class ProductController : Controller
{
    public IActionResult Update(string product_id, int quantity)
    {
        …
    }
}
```

## Chapter 2: Reusable Layout

### Question 1

What are the advantages of using reusable layout in ASP.NET MVC web applications?

#### Reusable layout

- Provide users with **consistent experience** as they navigate from page to page
- Layout may **include common UI elements**:
  - Header
  - Navigation Menu
  - Footer
- **Common HTML structures are also included** in layout:
  - Shared JS and CSS imports
- Layout **reduces duplicate code in views**.

### Question 2

What are tag helpers?

- Tag helpers **extend the functionality of HTML elements** via **supporting non-standard custom HTML attributes**.

- **Enable server-side** code to participate in **creating and rendering HTML elements in views**.
- Tag helpers are **processed at server-side** and **rendered into standard HTML elements and attributes** which browsers can understand.

## Question 3

Differentiate between tag helpers and HTML helpers.

| | Tag Helpers | HTML Helper |
|---|---|---|
| Writing format | Written as **HTML elements** with custom non-standard attributes | Written as **C# methods**. Work like function calls with parameters. |
| Readability | Codes are **more readable** as they resemble typical HTML syntax | Codes are **less readable** as HTML and C# codes are mixed together |
| Accessibility | Codes are **clean, concise and easier to understand** | Codes are usually **more verbose and slightly complicated** |
| User-friendliness | Friendly to **web designers** who know little about programming | Natural to **web developers** who familiar with programming syntax |
| Efficiency | **Efficient in performance** as they are compiled as part of the view | Introduce **minor overhead** as they are executed as C# codes |

## Question 4

What are partial views?

- Partial view is a **reusable UI component** used to **render a portion of a web page**.
- Commonly used to **break down large views** in smaller, manageable and **reusable parts**.
- Helps to **avoid redundant codes** by rendering shared content in multiple places across an application.
- Usually used together with AJAX programming to **support partial page reload**.

# Chapter 3: JavaScript and AJAX Programming

## Question 1

Discuss how JavaScript (client-side processing) is different from C# (server-side processing).

| | JavaScript | C# |
|---|---|---|
| Where it run? | Run locally at client (browser) | Run at server |
| Strength | Less powerful | More powerful |
| Accessibility to database | No direct access to database | Can direct access to database |
| Complexity of task | Suitable for simple tasks:<br>• Basic input validations<br>• Basic calculations<br>• Image preview<br>• Animations and visual effects | Suitable for complex tasks:<br>• Application security (login, logout)<br>• Perform CRUD on database |
| Security | Insecure. User can turn off or modify JavaScript easily. | Secure. User cannot directly access and modify the C# code. |
| Responsiveness | No roundtrip. Immediate response. | Has roundtrip as client has to send request to the server for data processing, then wait for the server to return response back to the client. Thus, it may cause delayed response. |

## Question 2

Define unobtrusive JavaScript.

- Unobtrusive JavaScript is a way of writing JavaScript codes, it separate JavaScript (behavior) from HTML (markup).
- It avoid the use of inline JavaScript from event handling.
- It register the event programmatically using DOM programming model.

## Question 3

The following event handling approach is not recommended. Suggest and discuss a better way to handle event in JavaScript.

NOTE: Code example is not required.

<button onclick=”alert(‘Hello’)”>Hello</button>

- Instead of using “onclick” attribute to trigger alert() JavaScript function, add an id attribute for the “button” element such as “hello-alert”.
- Then, attach event to document object, but targeting element with id attribute “hello-alert”.
- If the “click” event is triggered, call the alert(‘Hello’) function in the script.

## Question 4

Discuss the advantages of writing unobtrusive JavaScript.

| | |
|---|---|
| Maintainability | • All JS code is separated from HTML, making the client-tier easier to maintain and update.<br>• Making the code easier to read (less messy) |
| Task distribution | • Web designers can work on HTML and CSS without worrying how to program JS code<br>• Web developers can work on JS without affecting the works of web designers |
| Modularity | • Making JS modules plug-able and reusable<br>• Allow new JS modules to be added easily, without affecting existing JS modules and HTML |
| Graceful degradation | • Promote progressive enhancement, which ensuring a web app continue to work (on the basic functionality), although certain browser features are not supported.<br>进渐进增强，确保网络应用程序在不支持某些浏览器功能的情况下继续运行（基本功能）。 |

## Question 5

Discuss the problems of traditional page loading approach without AJAX.

- In traditional page loading approach, the **full-page reload** will always happen when a link is clicked or a form is submitted.

- Once the full-page is reloaded, the **view context will be lost** when the **scroll position will reset** and the **page goes back to top**.
- **More bytes are needed to download** when the static content is also reloaded.

## Question 6

Discuss AJAX programming technique and its advantages.

- AJAX (Asynchronous JavaScript and XML)
  - **Handle HTTP request and response using JavaScript** running in the background.
  - Allow form submission and content update **without reloading the whole page**.
  - **Only the updated portion is sent to client** (rather than whole page) for reducing bytes need to be downloaded.
  - **Only the targeted portion is refreshed** (rather than whole page) for retaining the view context of the users.
- Advantages:
  - Improve responsiveness
  - Improve user experience

## Question 7

Suggest some AJAX features that can be implemented into web applications.

- AJAX searching
- AJAX sorting
- AJAX paging

## Question 8

Discuss the AJAX polling technique.

- AJAX polling is a process of **continuously sending AJAX requests to server in background** to **obtain updated data or content**.
- AJAX requests are sent automatically, usually by a **fixed interval** (e.g. every 10 seconds), **without user interaction**.
- It can act as a simple technique to **simulate real-time communication** for features like real-time chat, price and update.

- It is **less efficient** as it might cause a **heavy load on both client and server** if the **request interval is small** such as every 1 second. (WebSocket, Server-Sent Events, WebRTC would perform better)

### Question 9

What is the problem of AJAX polling?

- AJAX polling only simulates real-time communication but it is not an actual real-time communication technique.
- If the request interval is small (e.g. 1 second), both the client and server will be heavily loaded.
- Less efficient compare to modern techniques (Server-Sent Events, WebSockets and WebRTC).

### Question 10

Discuss the purpose of extension methods in C#.

- Allows for **adding new methods** to existing types **without modifying the original types**.
- Helpful for **adding functionality to classes** where we **do not have access to the source code**. (encapsulation)
- Similar to **creating reusable functions** with the functions closely associated to the targeted classes or types.

## Chapter 4: Object-Relational Mapping (ORM)

### Question 1

What is object-relational mapping (ORM)? Identify how relational model (table, column, record and relationship) is mapped to object model.

- Object-Relational Mapping (ORM) is a process of **converting data in a relational database into object-oriented representations**.
- Mapping from relational model to object model:

| Relational Model | | Object Model |
|---|---|---|
| Table | map to | Class |

| | | |
|---|---|---|
| Column | | Property |
| Relationship | | Association |
| Record (Row) | | Object |

## Question 2

Discuss the advantages of using ORM tool (e.g. Entity Framework Core) from the following aspects:

- Automation
- Object-oriented
- SQL-less
- Database supports
- Cleaner, consistent and secure codes

- Automation
  - **Entity classes can be generated automatically from an existing database**. (database-first approach)
  - Or, a **database can be generated automatically from existing entity classes**. (code-first approach)
  - Without ORM tools, web developers have to program the entity classes manually which can be troublesome.
- Object-oriented
  - **Allow database records to be accessed using OOP approach**.
  - Deal with classes, objects, properties and associations.
- SQL-less
  - **Data access functions are built-in as properties or methods**.
  - **No need to write SQL** to perform CRUD operations.
- Database supports
  - **Support multiple databases from different vendors through provider model**.
  - E.g. SQL Server, MySQL, SQLite, Oracle.
- Clean, consistent and secure codes
  - **Programming codes are not mixed with SQL statements**.
  - Everything is **object-oriented**.
  - Easier to program, read and maintain.
  - **SQL injection issue can be handled** by ORM tools automatically.

## Question 3

Discuss the key disadvantage of using ORM tool.

- Performance
    - ORM tool usually **creates overhead** and **performs slower** as compare to embedding SQL statements directly in the codes.
    - This is because **objects need to be transformed into relational data** and vice versa
    - Complex queries that join multiple tables are usually not well-optimized by ORM tool.

## Question 4

Differentiate between database-first and code-first database development.

| | Database-First | Code-First |
|---|---|---|
| Development Sequence | **Develop database first**. Then generate entity classes automatically | **Develop entity classes first**. Then generate database automatically |
| Popular for whom? | **Database administrators** who focus on **database development** | **Programmers** who focus on **application development** |
| When it is suitable? | Suitable when **working with existing database** | Suitable when the **database has not been created** |
| Recommendation | Recommended for **large applications** that involve **extensive data processing** (manual DB optimization possible) | Recommended for **small to medium applications** that **do not involve extensive data processing** |
| Database type | Suitable for **shared and centralized database** that is **accessed directly by multiple applications or systems** | Suitable for **application specific database**, or when the **data is shared via Web API (indirect access)** |

## Question 5

Describe how services are made available to controllers via constructor dependency injection.

- The **services** (e.g. DbContext) that a **controller is depending on** will be **listed as parameters in the constructor**.
- The **framework** will **manage the services** (create and dispose) and **injecting them to the controller**.
- **Services to be injected are mapped by the class types or interface types specified in the constructor**.

- **Services need to be registered or added in Program.cs file** before they can be used in dependency injection.
- Some of the framework-registered services can be used directly without further configuration.

### Question 6

Discuss the reason and purpose of using database migration.

- Since **entities and properties are often being added and removed** during development, the **database scheme needs to be updated** accordingly.
- Database migration helps to **manage the incremental changes** and **keep the database scheme in sync**.
- It is especially useful in **code-first development** as it **allows database development without writing SQL scripts manually**.

## Chapter 5: CRUD Database Operations

### Question 1

What is the purpose of **Include()** method in a SELECT operation? NOTE: Code examples are not required.

Include() method can explicitly load the records from the related table into the result set.

### Question 2

We can SELECT a record from database by using **First()** or **FirstOrDefault()** method. Differentiate between them. NOTE: Code examples are not required.

- First()
  - If no record found, error will occur.
- FirstOrDefault()
  - If no record found, NULL will be returned.
  - Can do null checking.

## Question 3

What is the purpose of **Select()** method in a SELECT operation? NOTE: Code examples are not required.

Select() method is used for obtaining and returning the specific records from the specific database tables.

## Question 4

Differentiate between **GET** and **POST** request methods. Classify the SELECT, INSERT, UPDATE and DELETE operations to either **GET** or **POST** request methods respectively.

| | GET Request | POST Request |
|---|---|---|
| Function | To retrieve resource (data) from web server. | To create, modify or remove resource (data) from web server. |
| Action | For action that does not alter server state (data). | For action that alters server state (data). |
| Side effect | No side effect. | Has side effect. |
| Example | SELECT operation | INSERT, UPDATE and DELETE operations |

## Question 5

Identify if **GET** or **POST** request method is suitable for each of the following operations. State the reason clearly:

- Select a student record — GET
- Insert a student record — POST
- Update a student record — POST
- Delete a student record — POST
- Search a list of student records by name — GET
- Sort a list of student records by gender — GET
- Download a student profile — GET
- Upload a student profile photo — POST
- Download a PDF file — GET
- Upload a PDF file — POST

## Question 6

Examine the advantages of using the **ExecuteUpdate()** and **ExecuteDelete()** methods.

- It can update or delete one or multiple records **without pre-select the targeted records**.
- The **underlying SQL query is generated** and **sent directly to the database for execution**.
- **Less memory usage** as EF Core does not load the targeted records into memory and track for data changes.
- **Better performance** as only one SQL query (UPDATE or DELETE) is sent to the database. SELECT query is not required.

## Question 7

Examine the disadvantages of using the **ExecuteUpdate()** and **ExecuteDelete()** methods.

- The syntax is **more SQL-oriented** rather than object-oriented.
- The syntax may be **less readable** for UPDATE operation.
- **Do not check if targeted record exists or not** before execution.
- **Target a single table only**. Navigation properties do not work.
- **Do not work for complex scenarios** that need to **track the changes of parent and child records all together**.

# Chapter 6: Inputs and Forms

## Question 1

Differentiate between domain models and view models.

| | Domain Models | View Models |
|---|---|---|
| Scope | Also called as **entity classes** | Typical **C# classes** |
| Definition | Model classes that **map directly to database tables** | Model classes that **do not map directly to database tables** |
| Purposes | Mainly for **automated database tables generation** purposes | Mainly for **inputs and outputs purposes on web forms and views** |
| Characteristics | Properties are marked with data annotations for database **columns definition** purposes | Properties are marked with data annotations for **input validation** purposes |

## Question 2

Briefly explain the purpose of the following validation attributes. NOTE: Code examples are not required:

- [Required]
- [StringLength]
- [MinLength]
- [MaxLength]
- [EmailAddress]
- [Url]
- [Compare]
- [Range]
- [RegularExpression]
- [Remote]

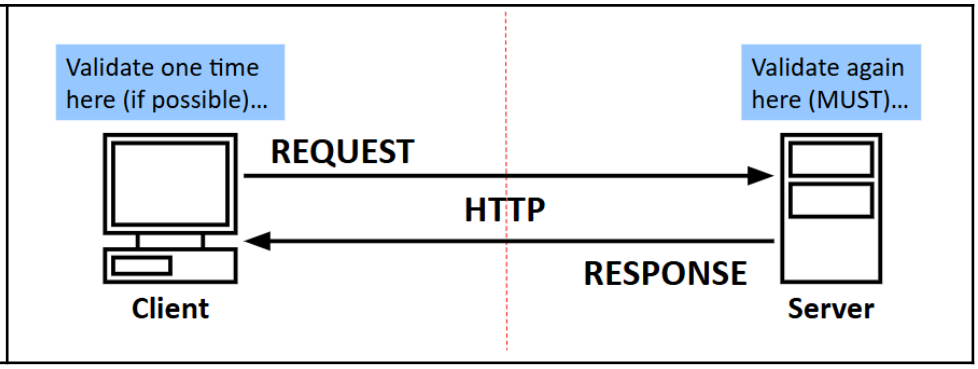
- [Required]
  - **Validate the property is filled (not null and not empty)**
  - No need to set in most cases, as non-nullable data types are required by default.
- [StringLength]
  - **Validate a string does not exceed the given maximum length.**
- [MInLength]
  - **Set the minimum length of a string**
  - **Set minimum number of items allowed in collection types (array and list)**
- [MaxLength]

        - **Set the maximum length of a string**
        - **Set maximum number of items allowed in collection types (array and list)**
- [EmailAddress]
    - **Validate if a string is a valid email address (test its format)**
- [Url]
    - **Validate if a string is a valid URL (test its format)**
    - The string should start with ftp://, http:// or https://
- [Compare]
    - **Validate if a value is same with another property**
    - Work with string, number and datetime
- [Range]
    - **Validate if a number falls between a given range**
    - **Minimum and maximum bounds are inclusive by default** (can set them to be exclusive manually)
- [RegularExpression]
    - **Validate if the input matches the given regular expression**
    - The pattern matches the entire input. As a result, the ^ (start of string) and $ (end of string)) symbols are optional
- [Remote]
    - **Send an AJAX request to remote server to check for input validity**
    - Only run at client-side (no server-side validation)

## Question 3

Differentiate between client-side and server-side input validations. Then recommend which one should be implemented into modern web systems.

| | Client-Side Input Validation | Server-Side Input Validations |
|---|---|---|
| Protection Layer Sequence | **First** layer of protection | **Second** layer of protection |
| Protection Layer type | **Sub** layer of protection | **Main** layer of protection |
| Programming Language | Program using **JavaScript run at browser (locally)** | Program using **C# run at server (remotely)** |
| Pros | • **Immediate response**<br>• **Improve responsiveness and user experience** | • **Access full programming resources**<br>• Capable to **perform complex validation logics** |
| Cons | **Incapable to perform complex validation logics** | **Involve roundtrip**, thus is **less responsive** |
| Security | **Insecure**, as JS can be turned off | **Secure**, server-side input validations always run |

| Diagram |  |
|---|---|

## Question 4

Differentiate between strongly-typed models and ViewBag properties.

| | Strongly-Typed Models | ViewBag Properties |
|---|---|---|
| Purposes | Pass data to view as an **object with its data type specified in the view** | Pass data to view as **dynamic properties of the ViewBag object** |
| Number of model accepted | A view can **only accept one model** | Capable to **pass multiple data to view** |
| Auto code completion | **Support** auto code completion | **Does not support** auto code completion |
| Design-time debugging | **Support** design-time debugging | **Does not support** design time debugging. |
| Page data type | Usually **hold main (primary) data** of the page | Usually **hold sub (secondary) data** of the page |

## Question 5

Illustrate with an example, describe how a controller pass (for captures) data to a view by using both strongly-typed model and ViewBag property. NOTE: Code examples are not required.

- A student registration form
- Student object is passed to view as strongly-type model to capture form inputs
- Program collection is passed to view using ViewBag to populate the dropdown selection list
- Since there are multiple data (Student object and Program collection), hence it is necessary to use both approaches at the same time for passing data

# Chapter 7: General File Handlings, Image Handlings, Reusable Services, Cookies and Sessions

## Question 1

Why it is not recommended to save an uploaded file using its original file name? Then, what file name should you use?

- If saving file using original file name, the **existing file with the same name will be overwritten**.
- The attacker may **inject folder path into the file name (path traversal attack)**. For example, ../../0001.txt will cause the file to be saved outside of the intended folder.
- I should **save the file using random unique file name**.
- I should implement the code which will **remove folder path** (if any) **from the given file name** and **reject malicious file name**.

## Question 2

Identify and discuss the THREE (3) image validation techniques with appropriate examples:

- Content Length
- Content Type
- File Extension

- Content Length
  - Ensure the **file size** is within the acceptable file size limit.
  - Example: 1 MB size  file size of 500 KB will be accepted, but file size of 2 MB will be rejected.
- Content Type
  - Ensure the file is an image by checking its **MIME type**.
  - Example: **image/jpeg** or **image/png**  file with MIME type equal to **image/jpeg** or **image/png** will be accepted, but file with other MIME type will be rejected.
- File Extension
  - Ensure the file is an image by checking its **file extension**.
  - Example: **.jpg** or **.png**  file with **.jpg** or **.png file** extension will be accepted, but file with other file extension will be rejected.

## Question 3

Identify and discuss the THREE (3) image processing techniques. Justify why each technique is important for web systems.

- Image Cropping
- Image Resizing
- Image Type Conversion

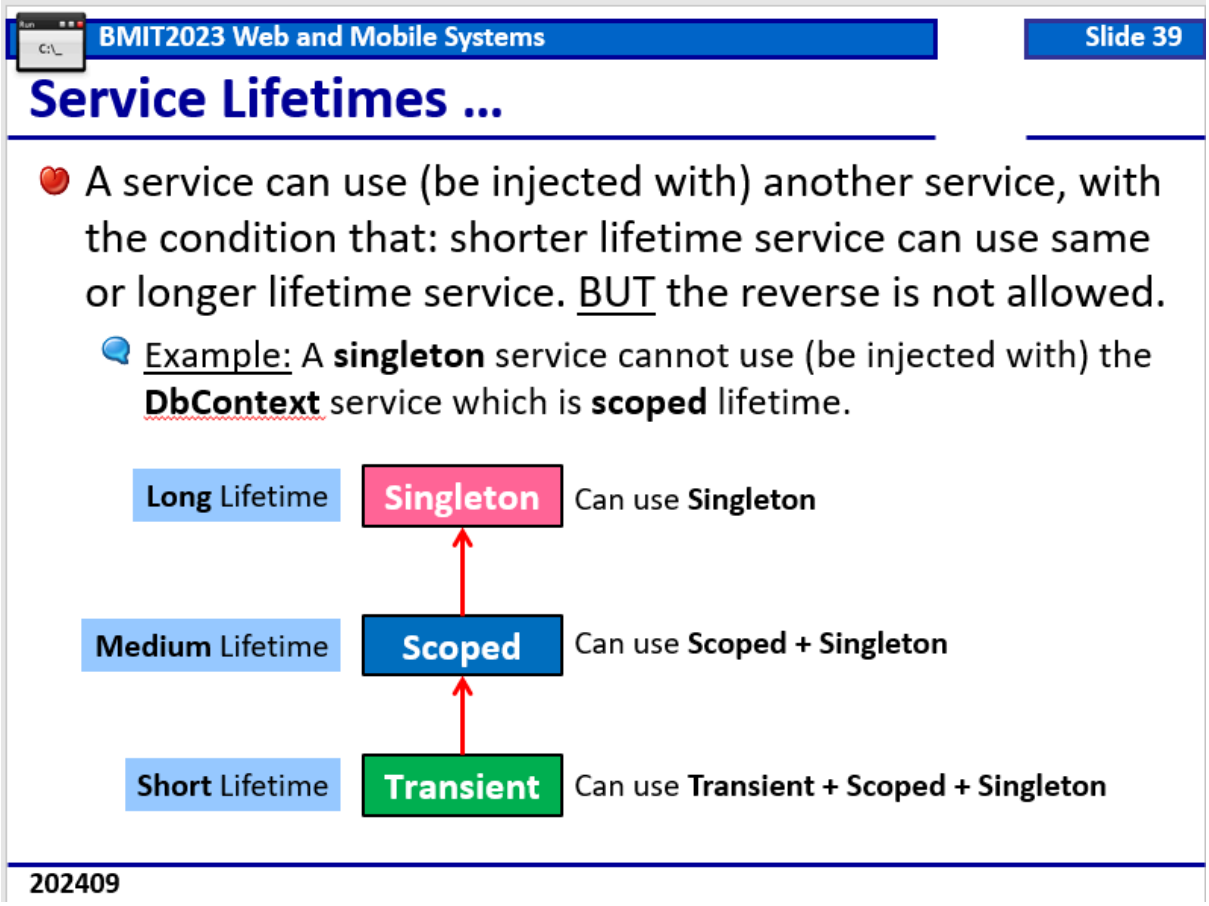
- Image Cropping
  - **Cut or drop the excessive portions of the image.**
  - To ensure all images are of the **consistent shape and aspect ratio**. Also **reduce file size**.
- Image Resizing
  - **Reduce the dimension (width and height) of the image.**
  - To ensure all images are of the **consistent dimension** (width and height). Also **reduce file size**, thus reduce storage space needed and download size.
- Image Type Conversion
  - **Convert images of different types to a single consistent type** (example: BMP, PNG, JPEG  JPEG)
  - To ensure all images are of the **consistent type**. **Easier file management**. Certain image type supports compression (JPEG)  smaller file size.

## Question 4

Identify and differentiate between the THREE (3) service lifetimes in ASP.NET Core. Specify the use case for each of them.

- Singleton
- Scoped
- Transient

| Aspect | Singleton | Scoped | Transient |
|---|---|---|---|
| Instance Count | Only one instance is created per application. | A new instance is created per HTTP request. | A new instance is created every time it is injected. |
| Lifetime | Live until the application stops or resets. | Live until the HTTP request-response ends. | Live until the calling method exits. |
| Reusability | Object instance created is shared globally. | Object instance created is shared within a request. | Not shared. New object instance every time. |
| Use Case | For services that need to be reused and shared across all HTTP requests, such as configuration, logging and caching. | For services that need to maintain consistent state within a single HTTP request such as EF Core's DB context. | For stateless and lightweight services that do not require shared state such as utility classes or builders. |




## Question 5

You want to add a service that retains and shares it states (data) globally across all pages and users. Which service lifetime should you use? Justify your answer.

- Singleton service lifetime
- A singleton service creates a single instance of the service when it is first requested.
- This instance is reused for all subsequent requests throughout the application's lifetime.
- Because the instance persists until the application stops or resets, it ensures that the state (data) can be consistently shared and retained across all pages and users.

## Question 6

You want to add a service that retains its states (data) within a single HTTP request and you also want to make use of DB context for database access within the service. Which service lifetime should you use? Justify your answer.

- Scoped service lifetime
- A Scoped service creates a new instance of the service for each HTTP request.
- The instance is retained and reused throughout the lifetime of the same request but is disposed of once the request completes.
- This ensures that a single DB context instance is used for all operations within a single HTTP request, which helps maintain consistency in data operations and prevents concurrency issues or unintentional data leaks between requests.

# Chapter 8: Web Security, Web Optimization, Software Monetization Models

## Question 1

Differentiate between authentication and authorization. For each of them, provide a use case example of how it can be implemented in a web system.

| Authentication | Authorization |
|---|---|
| Process of ascertaining 确认 the identify of a user. Ensure if the user is really the one who he claims to be. | Process of allowing or denying a user from accessing to protected pages, functions or resources. |
| Example:<br><br>Ask the user to provide a pair of username and password in login page. Authenticate the user if the username and password are matched with credentials stored in the database. | Example:<br><br>Associate each user with a role. Authorize the user for accessing protected resources based on his role. E.g. if he is “Admin”, he can add records, otherwise deny him from doing so. |

## Question 2

Examine what is cookie-based authentication.

- The most common form of authentication method for web systems. It is easy to implement and is widely supported.
- An authentication ticket (or token) is used to user after login, kept in the form of a secure, HTTP-only and encrypted cookie.
- User identify information is encrypted in the cookie. Such as user id (or email), role and other customizable claims.
- The cookie can be session-based (deleted when browser closes) or persistence (enable remember-me feature).
- The cookie is simply deleted when the user logout from the web system. There is no complicated process involved.

## Question 3

Refer to the codes given below. Analyse and discuss the effect of Line 01, 04 and 07:

```
01  [Authorize]
02  public IActionResult Page1() {...}
03
04  [Authorize(Roles = "Admin")]
05  public IActionResult Page2() {...}
06
07  [Authorize(Roles = "Admin, Member")]
08  public IActionResult Page3() {...}
```

- Line 01:
  - Only authenticated users (those who already login) can access this
- Line 04:
  - Only users with “Admin” role can access this
- Line 07:
  - Only users with “Admin” or “Member” role can access this

## Question 4

Differentiate between principal, identity and claims in relation to ASP.NET Core security.

- Principal
  - The **entity (human or machine) that interacts with the system**.
  - It can be a **user who is authenticated by email and password**.
  - It can be other **system that is authenticated by a secret key**.
- Identity
  - A **proof that indicates the user has been authenticated**.
  - **Contains user information (claims)** associated to the identity.
  - In a complex system, a user can have multiple identities.
- Claims
  - **Small pieces of user data**. Example: user id, email, role, etc.
  - Claims are usually **maintained as key-value pairs** internally.
  - **A group of related claims form an identity**.

## Question 5

Use a real-life analogy or metaphor to illustrate the differences between principal, identify and claims.

A simple real-life analogy or metaphor: