Asp.NET MVC Core

Trainer : Priyanka R Rangole

Email: priyanka.rangole@sunbeaminfo.com

Entity Relation Model

Entity-Relationship (ER) Diagram

- A **visual tool** to represent data and relationships in a database.
- Helps in designing databases clearly and efficiently.
- Shows **entities, attributes, relationships**, and **constraints**.

What is an ER-Diagram?

- A **pictorial representation of data**.
- Helps understand **how data is related**.
- Components:
    - **Entities** – real-world objects or things.
    - **Attributes** – details about entities.
    - **Relationships** – how entities interact or connect.

Entities

- Represented using **rectangles**.
- Named using a **singular noun**.
- Example entities:
  - Student
  - Course
  - Department
- **Purpose:** Represent real-world objects in a database.

Attributes

- Represented using **ovals** (in Chen notation).
- Describe **properties of entities or relationships**.
- Examples:
  - Student → Name, Roll Number, Age
  - Course → CourseID, Title, Credits
- Some attributes can be:
  - **Key attribute** (unique identifier, underlined)
  - **Composite attribute** (made of multiple sub-parts)

Relationships

- Represented using **diamonds**.
- Describe **how entities are connected**.
- Example:
  - Student **enrolls in** Course
  - Teacher **teaches** Course
- Types:
  - **One-to-One (1:1)**
  - **One-to-Many (1:N)**
  - **Many-to-Many (M:N)**

Entity Type and Occurrence

1.Entity Type:

1. A **category of similar objects** with the same properties.
2. Example: All students → Student entity type.

2.Entity Occurrence (Instance):

1. A **single object** of an entity type.
2. Example: Student “Rahul” → one occurrence of Student entity.

ER Modeling Approach

- **Top-Down Structure:**
  - Identify **main entities** and relationships.
  - Add **attributes and constraints**.
  - Define **primary and foreign keys**.
- Advantages:
  - Ensures **logical organization**.
  - Minimizes **design errors**.

Example ER Scenario

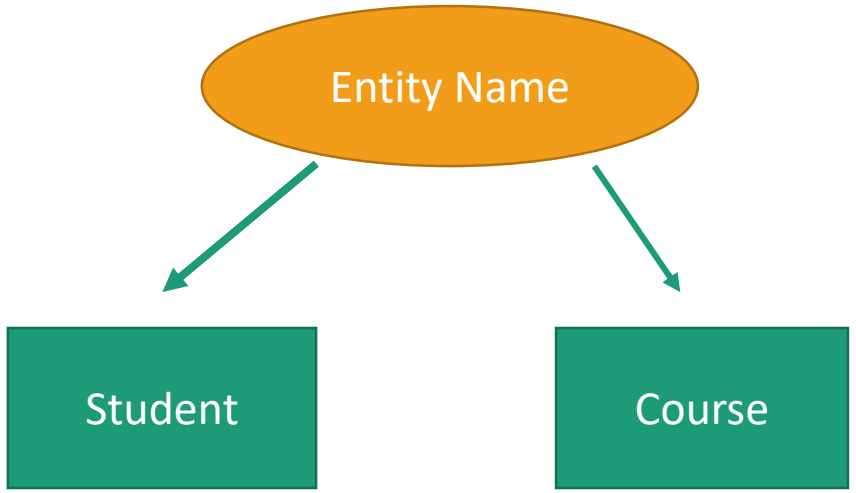
College System:

- Entities: Student, Course, Teacher, Department
- Relationships:
  - Student **enrolls in** Course (M:N)
  - Teacher **teaches** Course (1:N)
  - Course **belongs to** Department (1:N)
- Attributes:
  - Student → StudentID, Name, Age
  - Course → CourseID, Title, Credits

Relationship	Meaning
1:1	One-to-One → one entity of type A relates to exactly one of type B
1:N	One-to-Many → one entity of type A relates to many entities of type B
M:N	Many-to-Many → many entities of type A relate to many entities of type B

Diagrammatic Representation of Entity Type

Each entity type is shown as a rectangle labeled with the name of the entity, which is usually a singular noun.



Diagrammatical representation of Entity Student and Course

ER-Diagram: Relationship Types and Attributes

1. What is a Relationship Type?

- A **relationship type** shows how entities are connected in a database.
- It is a **set of associations** between **one or more entity types**.
- Each relationship is given a **name** that describes its function.
- Example: *"Branch has Staff"* → the name of the relationship is **"has"**.

2. Participants in a Relationship

- **Participants** are the entities that take part in a relationship.
- **Degree of a relationship** = number of participating entities.
    - **Binary relationship (degree 2)** → two entities involved
    - **Ternary relationship (degree 3)** → three entities involved
- Example: *Branch has Staff* → Branch and Staff are participants → **Binary relationship**

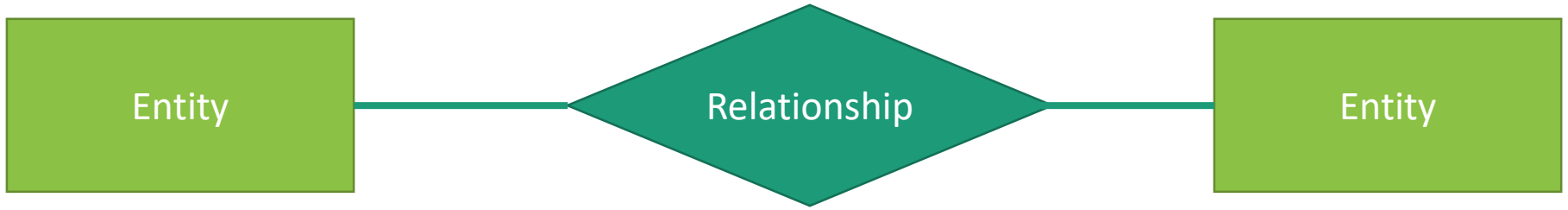
3. Attributes in ER Diagrams

- **Attributes** = properties or details of entities.
- Represented using **ellipses** connected to the entity rectangle.
- Example:
    - Student → Name, Roll Number, Age
- **Multi-valued attributes** → represented by **double ellipses**.
    - Example: Phone Numbers of a student

Representation of Relationships

4.

- **Diamond-shaped box** = Relationship
- Lines connect the **entities (rectangles)** to the relationship.
- Shows which entities participate in which relationship.



Types of Relationships

a) One-to-One (1:1)

•Only **one instance of entity A** is related to **one instance of entity B**.

•Example:

- *Each person has one passport*

b) One-to-Many (1:N)

•**One instance of entity A** is related to **many instances of entity B**.

•Example:

- *One department has many staff members*

c) Many-to-One (N:1)

•**Many instances of entity A** relate to **one instance of entity B**.

•Example:

- *Many employees belong to one branch*

d) Many-to-Many (M:N)

•**Many instances of entity A** relate to **many instances of entity B**.

•Example:

- *Students enroll in many courses; each course has many students*

Enhanced Entity-Relationship (EER) Model

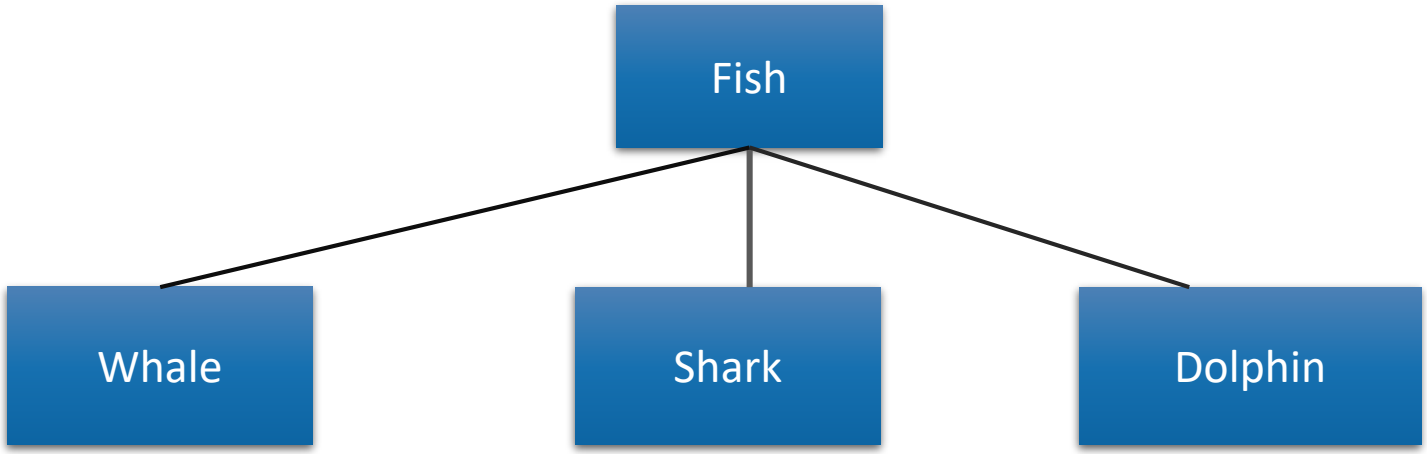
Why EER Model?

- Traditional ER modeling may not be enough for **complex modern applications**.
- Enhanced ER (EER) adds **semantic concepts** to handle advanced database requirements.
- Main added concepts:

1.Generalization / Specialization
2.Aggregation
3.Composition

Superclass/Subclass (Generalization):

In the below-mentioned figure, whale, shark, and dolphin are generalized as fish, i.e., they have been categorized as the fish.



Generalization / Specialization

Definition

- **Generalization:** Bottom-up approach
  - Combines multiple similar entities into a **generalized entity** (superclass).
  - Example: Whale, Shark, Dolphin → generalized as **Fish**.
- **Specialization:** Top-down approach
  - Divides a general entity into **specialized subgroups** (subclasses).

Key Terms

- **Superclass:**
  - General entity that contains one or more sub-groups.
  - Example: Fish
- **Subclass:**
  - Distinct subgroup of a superclass based on specific characteristics.
  - Example: Whale, Shark, Dolphin

Rules

- Every **subclass member is also a member of the superclass**.
- **Relationship between superclass and subclass** is **1:1**.
- Subclass may have **additional attributes** specific to it.

Aggregation

Definition

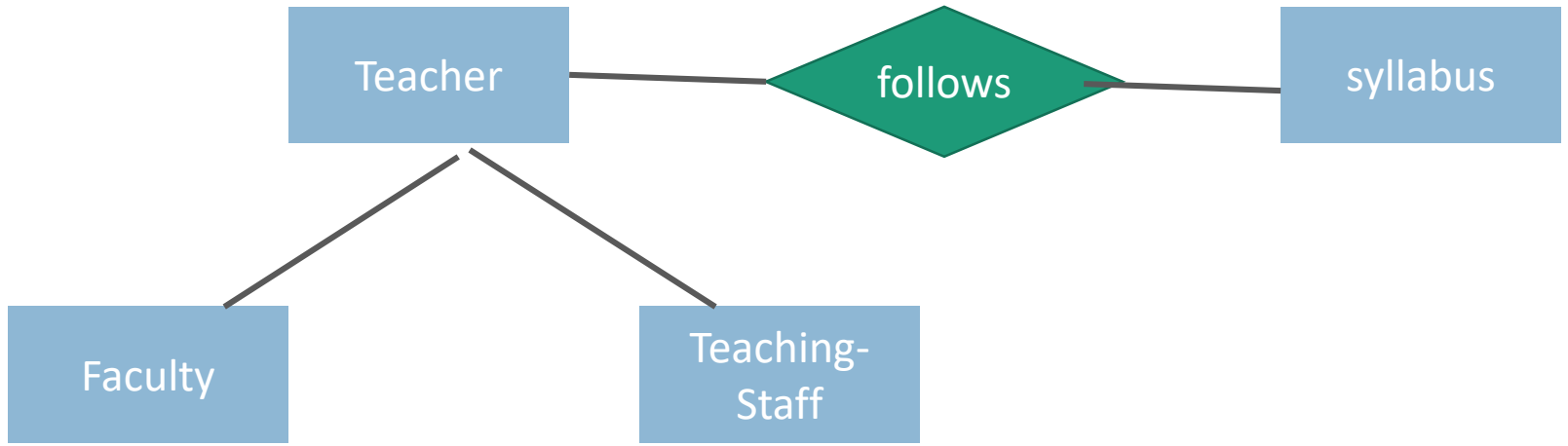
- Aggregation is a process where a **relationship between two entities** is treated as a **single higher-level entity**.
- Useful for modeling **"has-a", "is-a", or "is-part-of"** relationships.
- Does **not change navigation or connection** across entities.

Example

- Entities: Teacher and Syllabus
- Aggregation: Treat **(Teacher + Syllabus)** as a **single entity** for another relationship.
- Allows representing **complex relationships as one unit**.

Key Point

- Aggregation is about **conceptual grouping** of entities and relationships to simplify modeling.



Concept	Approach	Meaning	Example
Generalization	Bottom-up	Combine entities with similar characteristics into a general entity	Whale, Shark, Dolphin → Fish
Specialization	Top-down	Divide a general entity into subgroups	Fish → Whale, Shark, Dolphin
Superclass	-	General entity containing subgroups	Fish
Subclass	-	Specific entity within a superclass	Whale
Aggregation	Conceptual grouping	Treat a relationship as a single entity	Teacher + Syllabus

Application Types

What is an Application?

•An **Application** is a program that contains a set of instructions for the CPU
•It performs specific tasks for the user
•Applications can be developed using different programming languages

Types of Applications

There are generally **two types of applications**:
1.Native Application
2.Web Application

Native Application

•Developed using languages like **C, C++** (also Java, C# etc.)
•Installed directly on the operating system
•**Faster performance**
•**OS dependent** (Windows, Linux, Android, iOS specific)
Examples:
•MS Word (Desktop)
•VLC Media Player

Web Application

•Developed using **HTML, CSS, JavaScript**
•Runs inside a **web browser**
•**Slower than native applications** (depends on network)
•**OS independent**
Examples:
•Gmail
•Facebook
•Online banking systems

Web Architecture (Overview)

Web Architecture explains how:
•Client communicates with server
•Requests are processed
•Responses are returned
Main Components:
1.Web Server
2.HTTP Request
3.HTTP Response

Web Server
•A **Web Server** delivers web content to clients
•Handles requests from browsers or apps
•Uses **HTTP / HTTPS protocols**

Popular Web Servers
1.Apache HTTP Server
1. Open-source 2.Highly flexible

2.Nginx
1. High performance 2.Handles large traffic

3.Microsoft IIS
1. Windows-based 2.Integrated with ASP.NET

Client–Server Architecture

- Fundamental architecture used in modern web applications
- Widely used in web, mobile, and enterprise systems

What is Client–Server Architecture?

- Client-Server architecture is a **network model**
- Multiple **clients** request services from a central **server**
- Communication follows **request–response** mechanism
- Commonly used in web applications, databases, email systems

Client

- Client is a **requester of services**
- Examples:
  - Web browser
  - Mobile application
- Responsibilities:
  - User interface
  - Sending requests to server
  - Displaying server responses

Server

- Server is a **service provider**
- Handles:
  - Business logic
  - Data processing
  - Database access
- Sends responses back to clients
- Can serve **multiple clients simultaneously**

Request–Response Flow

1. User performs an action on client (browser/app)
2. Client sends request to server
3. Server processes the request
4. Server sends response back
5. Client displays result to user

Server

What is a Server?

- A **Server** is a physical machine with high configuration
- Processes large-scale data
- Sends responses to multiple clients

Server Components

A server generally consists of:

1. Web Server
2. Database Server
3. Programming Languages
4. Operating System (Platform)

Software Stack

- A **Software Stack** is a collection of software used together
- Defines how applications are developed and hosted

WISA Stack

- **W** – Windows
- **I** – IIS
- **S** – SQL Server
- **A** – ASP.NET

Used for:

- ASP.NET / ASP.NET Core applications

MEAN Stack

- **M** – MongoDB
- **E** – Express.js
- **A** – Angular
- **N** – Node.js

Used for:

- Full-stack JavaScript applications

HTTP Request

•Sent by **client** to **server**
•Used to request a resource or perform an action
Main Parts:
1.Header
2.Body (optional)

HTTP Request – Header
Headers provide additional information
Examples:
•**Host**: example.com
•**User-Agent**: Browser / App info
•**Content-Type**: application/json

HTTP Request – Body
•Contains data sent to the server
•Used mainly in **POST** and **PUT** requests
Examples:
•Form data
•JSON data

HTTP Response

•Sent by **server** to **client**
•Contains:
- Status code
- Headers
- Body (optional)

HTTP Response – Header
Provides metadata about response
Examples:
•**Content-Type**: text/html, application/json
•**Content-Length**: Size of response
•**Set-Cookie**: Sends cookies

HTTP Response – Body
Contains actual response data
Examples:
•HTML page
•JSON data (API response)

HTTP Request Methods

•**GET** – Fetch data
PUT – Update data
POST – Send data
DELETE – Remove data

HTTP Response Status Codes

1xx – Informational
•100 Continue

2xx – Success
- 200 OK
- 201 Created

HTTP Response Status Codes (Contd.)

3xx – Redirection
•301 Moved Permanently
•302 Found

4xx – Client Errors
•400 Bad Request
•401 Unauthorized
•404 Not Found

5xx – Server Errors
•500 Internal Server Error
•503 Service Unavailable

Key Components

- **Client** – Sends request
- **Server** – Processes request
- **Network** – Connects client and server
- **Protocol** – Rules of communication (HTTP/HTTPS)
- **Database** – Stores data
- **Middleware** – Authentication, logging, validation

Why Client–Server Architecture?

- **Centralized control** of data and services
- **Easy maintenance** and updates
- **Scalable** – supports many users
- **Better security** through centralized access
- **Efficient resource usage**

Types of Client–Server Architecture

1-Tier

- Client, server, and database on same machine => Example: Standalone desktop application

2-Tier

- Client ↔ Server => Example: Client app directly accessing database

3-Tier

- Client ↔ Application Server ↔ Database => Most common in web applications

Client-Side Processing

- Handles:
  - UI rendering
  - User interaction
- Technologies:
  - HTML
  - CSS
  - JavaScript
- Examples:
  - Form validation
  - UI updates

Server-Side Processing

- Handles:
  - Business logic
  - Authentication
  - Database operations
- Technologies:
  - ASP.NET
  - Java (Spring)
  - Node.js
  - Python (Django)

Communication Protocols

- **HTTP / HTTPS** – Web communication
- **TCP/IP** – Network communication
- **REST APIs** – Client-server data exchange
- **WebSocket's** – Real-time communication

Disadvantages

- Server failure can affect all clients
- Network dependency
- Higher setup and maintenance cost
- Requires skilled administration

Advantages

- High **scalability**
- **Centralized data management**
- Improved **security**
- Better **performance** with caching and load balancing
- Easy **maintenance and upgrades**

Real-World Examples

- Web applications (Google, Facebook)
- Banking systems
- Online shopping systems
- Email services
- Enterprise applications

Why MVC Fits Client–Server?

- Clear separation of UI, logic, and data
- Easy maintenance and scalability

.NET Core Overview

What is .NET Core?
.NET Core is a **free, open-source, general-purpose development platform** maintained by Microsoft.
It is a **cross-platform framework** that runs on **Windows, macOS, and Linux**.
.NET Core is a modern replacement for the traditional **.NET Framework**, designed to overcome its limitations and support today's application needs.

What Can We Build Using .NET Core?
Using .NET Core (now called **.NET**), we can build:
- Web applications
- Web APIs
- Desktop applications
- Mobile applications
- Cloud-based applications
- Microservices
- IoT applications
- Machine Learning apps
- Games

Why .NET Core Was Introduced?

The traditional **.NET Framework** had several limitations:

- Works only on **Windows**
- Different APIs for:
  - Desktop
  - Web
  - Windows Store
  - Windows Phone
- Installed **machine-wide**
  - Changes affect all applications

Today, applications usually include:

- Web backend
- Desktop admin panel
- Web & mobile client apps

👉 A **single, unified, cross-platform framework** was needed.

To solve this, Microsoft introduced **.NET Core**.

Key Design Goals of .NET Core

.NET Core was designed to be:
•Cross-platform
•Open-source
•Modular
•Lightweight
•High performance
•Easy to maintain

Modular Architecture
.NET Core is built **from scratch** using a **modular approach**.
•Core runtime contains only essential features
•Additional features are added using **NuGet packages**
•Developers include **only what is needed**

Benefits:
•Faster application performance
•Reduced memory usage
•Easier maintenance
•Smaller deployment size

.NET Version History

Important Note on Naming

- **.NET Core 3.1** was the last version named ".NET Core"
- From the next release, it was renamed to **.NET 5**
- Current versions are simply called **.NET**

.NET Version History

Version	Release Date	Support
.NET 7	Nov 8, 2022	STS
.NET 6 (LTS)	Nov 9, 2021	Nov 12, 2024
.NET 5	Nov 10, 2020	Ended
.NET Core 3.1 (LTS)	Dec 3, 2019	Ended
.NET Core 2.1 (LTS)	May 30, 2018	Ended
.NET Core 1.0	Jun 27, 2016	Ended

LTS (Long-Term Support) versions are recommended for production.

Unification of .NET Platform

Starting with **.NET 5**, Microsoft unified:

- .NET Core
- .NET Framework
- Xamarin

into a **single platform called ".NET"**.

This allows developers to:

- Learn one framework
- Use consistent APIs
- Build multiple application types

Key Characteristics of .NET Core / .NET

1.Open Source
- Maintained by Microsoft
- Available on GitHub
- Licensed under MIT and Apache 2.0
- Part of the **.NET Foundation**

2.Cross-Platform
- Runs on Windows, macOS, Linux
- Separate runtime for each OS
- Same code → same output

3.Architecture Independent
- Supports:
  - x86
  - x64
  - ARM

4.Supports Multiple Languages
- C#
- F#
- Visual Basic

IDEs:
- Visual Studio
- Visual Studio Code
- Sublime Text
- Vim

5.CLI Tools

.NET provides powerful **command-line tools**:
- Project creation
- Build
- Run
- Publish
- CI/CD support

6.High Performance
- Optimized runtime
- Uses **JIT (Just-In-Time) compilation**
- Faster execution and better resource usage

7.Compatibility
- Compatible with:
  - .NET Framework
  - Mono
- Uses **.NET Standard** for API compatibility

8.Flexible Deployment
- Framework-dependent deployment
- Self-contained deployment
- Docker container support

Useful Links

- Official Documentation:
https://learn.microsoft.com/dotnet/
- Download SDK & Runtime:
https://dotnet.microsoft.com/download

ASP.NET Core Overview

What is ASP.NET Core?

ASP.NET Core is a **modern, completely re-written version** of the traditional ASP.NET web framework.

It is a **free, open-source, high-performance, and cross-platform framework** used to build **cloud-based and internet-connected applications**.

Using ASP.NET Core, we can build:

- Web Applications
- Web APIs
- Mobile backends
- Cloud-based applications
- IoT applications

ASP.NET Core is designed to run:

- **On the cloud**
- **On-premises servers**

Modular and Lightweight Architecture

Like .NET Core, ASP.NET Core is **modular by design**.

- Core framework contains only essential components
- Advanced features are added using **NuGet packages**
- Developers include only what is required

Benefits:

- High performance
- Less memory consumption
- Smaller deployment size
- Easier maintenance

Why ASP.NET Core?

ASP.NET Core provides many advantages over the traditional ASP.NET framework.

Cross-Platform Support
ASP.NET Core applications can run on:
•Windows
•Linux
•macOS
👉 No need to build separate applications for different platforms.

Built-in IoC (Dependency Injection)
ASP.NET Core includes a **built-in Inversion of Control (IoC) container**.
Benefits:
•Automatic dependency injection
•Loose coupling
•Easy testing
•Better maintainability

High Performance
•ASP.NET Core does **not depend on System.Web.dll**
•Lightweight request pipeline
•Minimal overhead
This results in:
•Faster execution
•Better scalability
•Improved performance

Integration with Modern UI Frameworks
ASP.NET Core works well with modern front-end frameworks such as:
•Angular
•React
•Vue
•Bootstrap
👉 Front-end and back-end can be developed independently.

Flexible Hosting

ASP.NET Core applications can be hosted on:

•IIS (Windows)

•Apache

•Nginx

•Docker containers

•Cloud platforms

👉 It is **not limited to IIS**, unlike traditional ASP.NET.

Smaller Deployment Footprint

•ASP.NET Core runs on **.NET runtime**

•Applications include only required libraries

Result:

•Smaller deployment size

•Faster startup time

•Reduced server load

Code Sharing

ASP.NET Core allows:

•Creating reusable class libraries

•Sharing code with:

- .NET Framework
- Mono
- Other .NET applications

This helps maintain a **single codebase** across platforms.

Side-by-Side Versioning

ASP.NET Core supports **multiple application versions running simultaneously**.

Benefits:

•No breaking existing apps

•Easy upgrades

•Better version control

•ASP.NET Core comes **bundled with .NET**

•No separate installation required for ASP.NET Core

ASP.NET Core is a **web framework built on top of .NET**, and it follows **middleware + MVC architecture**

Cross-Platform ASP.NET Core Architecture

ASP.NET Core applications run on:
•Windows
•macOS
•Linux
ASP.NET Core applications are **self-hosted** using an internal web server called **Kestrel**.

```
Client
  ↓
External Web Server (IIS / Nginx / Apache)
  ↓
Kestrel Web Server
  ↓
ASP.NET Core Application
```

👉 External web servers forward requests to **Kestrel**, which processes them.

ASP.NET Core Request Handling
ASP.NET Core uses a **middleware pipeline** to process HTTP requests.
•Each middleware performs a specific task
•Requests pass through middleware sequentially
•Responses travel back in reverse order

ASP.NET Core Version History

- ASP.NET was first introduced in **2002** with .NET Framework
- It worked only on **Windows**
- In **2016**, Microsoft introduced **ASP.NET Core**
- ASP.NET Core runs on Windows, macOS, and Linux

Naming Change After ASP.NET Core 3.1

- **ASP.NET Core 3.1** was the last version with the "Core" name
- Next version was released as **ASP.NET 5**
- From then onwards, it is simply called **ASP.NET**

👉 ASP.NET 5 and later versions are still **ASP.NET Core internally**

ASP.NET Core Versions

Version	Visual Studio	Release Date	End of Support
ASP.NET 7	VS 2022 v17.4	Nov 8, 2022	May 14, 2024
ASP.NET 6 (LTS)	VS 2022	Nov 9, 2021	Nov 12, 2024
ASP.NET 5	VS 2019	Nov 10, 2020	May 10, 2022
ASP.NET Core 3.1 (LTS)	VS 2019	Dec 3, 2019	Dec 13, 2022
ASP.NET Core 2.1 (LTS)	VS 2017/2019	May 30, 2018	Aug 21, 2021
ASP.NET Core 1.0	VS 2017	Jun 27, 2016	Jun 27, 2019