# Report

## 1. .NET Versions

### Key Versions

1. .NET Framework (2002 - Present):
    - The original version of .NET designed for Windows.
    - Supports web (ASP.NET), desktop (WinForms/WPF), and services.
    - Includes versions like 1.0, 2.0, 3.5, 4.x.
2. .NET Core (2016 - 2020):
    - A cross-platform, open-source framework for building modern applications.
    - Modular and lightweight, making it suitable for cloud-based and containerized applications.
    - Includes versions like .NET Core 1.0, 2.x, and 3.x.
3. .NET 5, 6, 7, and beyond (2020 - Present):
    - Unified framework that merges the features of .NET Core and .NET Framework.
    - Focuses on performance, simplicity, and cross-platform support.
    - .NET 6 is a Long-Term Support (LTS) version, with subsequent versions introducing enhancements.
4. **.**NET 8 (November 2024.)
    - Type: Long-Term Support (LTS).
    - Overview: introduced in .NET 7, with a focus on scalability, cloud-native development, and modern application paradigms.
    - Language Features(C#12).
    - Enhanced Just-In-Time (JIT) compilation.

## 2. Namespace

Overview
A **namespace** in .NET is a logical grouping of classes, interfaces, enums, and other types. It helps organize code and avoid name conflicts.

### Key Features

- **Organization:** Helps categorize related functionality .
- **Avoid Conflicts:** Prevents naming collisions by providing a fully qualified name .
- **Reusability:** Facilitates code reuse by logically separating functionality.

## 3. .NET Core

### Overview

.NET Core is a modular, high-performance, and cross-platform framework introduced by Microsoft in 2016. It is designed for modern application development, supporting Windows, macOS, and Linux.

### Features

- **Cross-Platform:** Run applications on multiple operating systems.
- **Open Source:** Actively developed and maintained by Microsoft and the community on GitHub.
- **High Performance:** Optimized for modern workloads.
- **Containerization:** Ideal for Docker and Kubernetes environments.
- **Unified Framework:** Starting from .NET 5, it merged with the .NET Framework to provide a single platform.

### Use Cases

- Web applications (ASP.NET Core).
- Microservices architecture.
- Cloud-based solutions.
- Cross-platform development.

## 4. Solution

### Overview

A solution in .NET is a container for managing multiple related projects in a structured way using Visual Studio or other IDEs.

### Key Features

- Organized Development: Groups projects (e.g., libraries, APIs, front-end) under one solution.
- Dependency Management: Simplifies referencing shared code across projects.
- Scalability: Supports small to enterprise-level applications with multiple layers (e.g., UI, business logic, data access).

### Components

1. Solution File (.sln):
    - A configuration file that stores information about the projects in the solution.
2. Projects:
    - Each project represents a specific part of the application (e.g., a class library, a web app).

# LinkedIn

**Basmala Said** • You
CS Student | Front-end Developer | React
5m •

الحكاية الطريفة بين .**NET Framework** والتجميع قبل وبعد
تخيل أنك في سباق سيارات، وكلما أردت أن تزيد سرعتك، تحتاج إلى التوقف لتعدّل المحرك.
هذا بالضبط ما كان يحدث في عالم **Net Framework**.! التجميع (**Compilation**) وقت التشغيل كان يشبه تلك التعديلات المستمرة. أما مع ظهور .**NET Core** و .**5 NET**+، الأمور تغيرت تمامًا، وأصبح لديك سيارة مجهّزة بالكامل منذ البداية، جاهزة للانطلاق بأقصى سرعة دون توقف!

دعنا نأخذك في رحلة ممتعة لنكتشف الفرق بين التجميع في الماضي والحاضر 🚗⚙️

---

التجميع في عالم .**NET Framework**: "أسرار الغموض"

كان الأمر أشبه بمشاهدة ساحر في السيرك. الكود الذي تكتبه بلغة مثل **C#** يتم ترجمته إلى **IL** (**Intermediate Language**)، ولكن هنا تبدأ المغامرة:

عندما تطلق برنامجك، يأتي دور الـ (**Common Language Runtime**) **CLR** ليقوم بترجمة الـ **IL** إلى لغة الآلة باستخدام تقنيته الفريدة (**JIT**) **Just-In-Time**.
هذا يعني أنه كل مرة تستدعي فيها دالة جديدة، يقول الـ **JIT**: "ثواني بس، خليني أترجم ده الأول!"

🪣 المشكلة؟

في كل مرة يتم تشغيل التطبيق، هناك وقت تأخير (**Overhead**) لأن الـ **JIT** يحب أن "يأخذ وقته".
إذا كان التطبيق كبيرًا، فقد تشعر وكأنك تشاهد فيلمًا بوليسيًا طويلًا ينتقل بين مشاهد التحقيق والاعترافات!

---

ثم جاء العصر الجديد: "**Pre-JIT** إلى الإنقاذ"
مع ظهور .**NET Core** و .**5 NET**+، قرر أبطال مايكروسوفت كسر هذا النمط. قالوا:
"ليه نخلي البرنامج يترجم كل حاجة في وقت التشغيل؟ ما نحضر العدة كلها قبل العرض!"
وهنا وُلدت فكرة **Ahead-of-Time** (**AOT**) **Compilation**، أو كما أسميها أنا: "تجميع قبل العرض". 🎬

مع **AOT**، يتم تحويل الكود إلى لغة الآلة قبل وقت التشغيل.
بمعنى آخر، أصبح التطبيق جاهزًا مثل "سندويتش جاهز للتقديم". 😋

---

💡 الملحوظة القلبوظة:

إذا كنت من عشاق الكود السريع والتطبيقات المتقدمة، اركب قطار .**NET Core**.
أما إذا كنت تحب الكلاسيكيات ولا مانع لديك من الانتظار قليلًا، فإن .**NET Framework** سيظل صديقك الوفي.