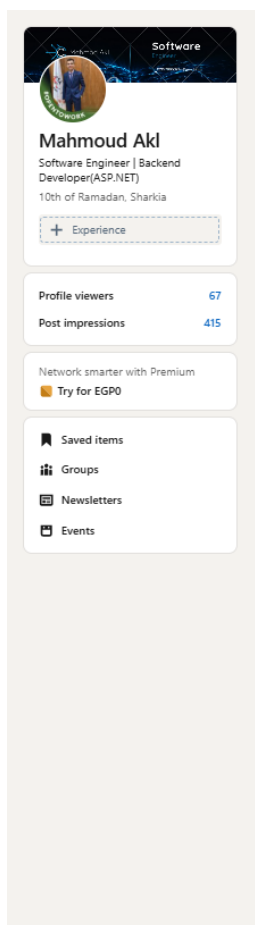


1) LinkedIn article about Compilation With dotnet FW Vs Pre

dotnet



الكود ده بتاع ويندوز بس! (Platform Dependency): لما كنت بتكتب كود بلغة زي ++C مثلاً، كنت بتعمله Compilation عشان يشتغل على نظام تشغيل معين (زي ويندوز) ومعالج معين (زي إنتل x86). يعني الكود اللي عملته لو ويندوز، مينفعش يشتغل على لينكس، والعكس صحيح. ده كان بيخلي نقل البرامج من جهاز لجهاز أو من نظام لنظام صعب جداً. كل لغة وليها قاعدتها (Lack of Language Standardization): كل لغة برمجة (زي ++C، ديلفي، فيجوال بيرك 6) كان ليها ال Compiler بتاعها اللي بيطلع ملفات تنفيذية بشكل مختلف. كان صعب أوي إنك تخلي برنامج معمول بلغة يكلم برنامج ثاني معمول بلغة ثانية. كنت محتاج حاجات معقدة زي ال COM عشان تعمل كده. أنت مسؤول عن كل حاجة حتى الذاكرة! (Manual Memory Management & Security Risks): في لغات زي ++C، المبرمج كان لازم هو اللي يقول للكمبيوتر "خد الحجة دي من الذاكرة"، وبعدين "أنا خلصت، رجع الحجة دي". لو نسيت ترجع الذاكرة دي، كان بيحصل حاجة اسمها Memory Leak. والبرنامج بيضوط أو بيتهج. كمان موضوع الأمان كان أصعب شوية، لأن الكود كان قريب أوي من الوصول لموارد الجهاز بشكل مباشر. مع ال .NET Framework: عصر جديد كله نظام وراحة! لما مايكروسوفت طلعت ال .NET Framework، قدمت طريقة جديدة لل Compilation قليت الموارد: خطوتين لل Compilation مش خطوة واحدة! (Two-Stage Compilation): الخطوة الأولى: يتحول ال (Common Intermediate Language) CIL: لما بتكتب كود بأي لغة من لغات ال .NET (زي VB.NET، C#)، الكود ده مش بيتترجم للغة الآلة على طول، لآ بيتترجم للغة وسيطة اسمها CIL أو MSIL. اللغة دي شبه لغة التجميع بس أعلى منها شوية، والأهم إنها مينعتمدش على أي نظام تشغيل معين. الخطوة الثانية: ترجمة فورية مع ال JIT (Just-In-Time Compilation): لما بتيجي تشغل البرنامج، بييج دور ال (Common Language Runtime) CLR، وده ري الموتور اللي بيشغل برامج ال .NET. ال CLR ده بيحول كود ال CIL للغة الآلة "في نفس اللحظة" اللي الكود ده هيشغل فيها. وده سر المرونة في ال .NET. لغة وسيطة واحدة لكل اللغات (Unified CIL): كل لغات ال .NET المختلفة بقت بتترجم لنفس ال CIL. ده معناه إن مكونات مكتوبة بلغة #C ممكن تشغل وتتفاهم بسهولة مع مكونات مكتوبة بلغة VB.NET أو #F. ده فتح الباب على مصراعيه لإننا نستخدم الكود بتاعنا أكثر من مرة وبنيني برامج ضخمة بأجزاء مختلفة من لغات مختلفة من غير أي مشاكل. إدارة الذاكرة أوتوماتيك والأمان بقي أحسن بكثير (Enhanced Security & Automatic Memory Management): دي وظيفة ال Garbage Collection. دي وظيفتها إنها تدير الذاكرة لوحدها، يعني المطور مش محتاج يشغل دماغه بحجز الذاكرة وتحريرها. ده قلل بشكل كبير جداً من أخطاء الذاكرة والبرامج اللي بتتهج. وكمان ال CLR بيوفر بيئة تشغيل آمنة، لأنه بيتأكد إن الكود سليم قبل ما يشغله، وبيتحكم في وصوله لموارد الجهاز، وده زود أمان البرامج بشكل عام. اشتغل في أي مكان (Platform Independence): بما إن الكود بيتترجم الأول ل CIL، فملف البرنامج (اللي هو ال DLL أو EXE) اللي فيه ال CIL ده، ممكن يشتغل على أي نظام تشغيل عليه ال CLR (زي ويندوز، وبعدين لينكس و macOS مع ظهور ال .NET Core). المفهوم ده اسمه "Write Once, Run Anywhere" يعني "اكتب مرة واحدة، شغله في أي مكان".



2) What has been done to reduce this overhead?

1) Tiered Compilation

- Starts by compiling methods quickly (Tier 0).
- frequently used methods are recompiled with optimizations (Tier 1).

2) ReadyToRun (R2R)

- Pre-compiles IL into native code during publish.
- Assemblies contain both IL and native code.

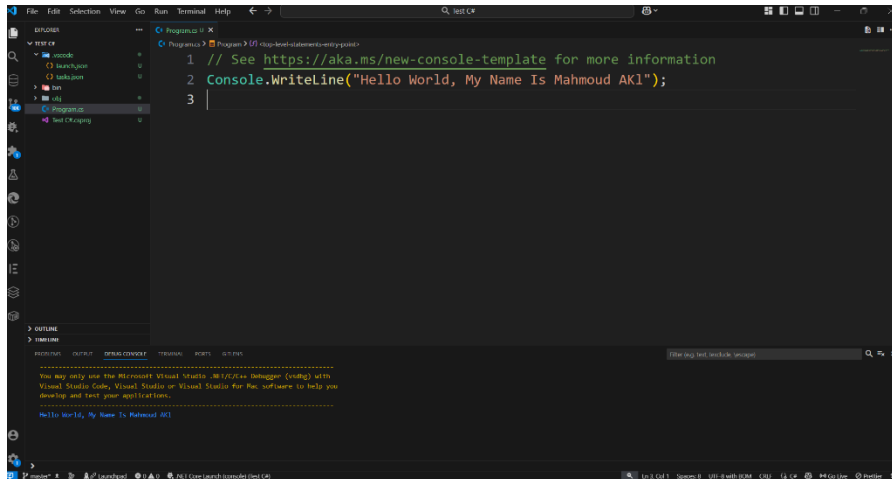
3) Native AOT (Ahead-of-Time Compilation)

- Compiles the entire application to native code before runtime.
- No JIT needed at all during execution.

4) CrossGen / CrossGen2 (Older method before Native AOT)

- Precompiles methods to native code at build time.
- Used by ReadyToRun internally.

3)



4)

Dot .NET Versions

- **.NET Framework:** مثال: Windows 4.8 بدأ من سنة 2002، بيشتغل بس على
- **.NET Core:** Multi-platform (Windows, Linux, macOS) نسخة خفيفة ومفتوحة المصدر. أول إصدار في 2016
- **.NET 5/6/7/8:** في 2020 .NET 5 فقط، ودي الموحدة لكل الأنواع. بدأت من ".NET" تسمى
- (دعم طويل الأجل – LTS) .NET 8. حاليًا: آخر إصدار

Namespace

- هي طريقة لتنظيم الكود داخل Projects.
- زي "مجلدات" داخلية Classes، Function، Interfaces.
- تمنع تعارض الأسماء بين Classes.

```
namespace MyApp.Services
{
    class Getdata { ... }
    class UserName { ... }
    class password { ... }
}
```

NET Core

- نسخة خفيفة وسريعة من .NET.
- + Multi-platform مفتوحة المصدر
- أداء أعلى من .NET Framework
- مثالي لتطبيقات الويب والـ APIs
- من 2020 بقى فيه دمج في اسم واحد .NET، يعني (Unified .NET = .NET Core + Framework).

4 Solution

- الـ Solution هو الملف الرئيسي .sln في أي مشروع C# أو .NET.
- يجمع مشاريع مختلفة (Projects) تحت مشروع واحد كبير
- ييسر إدارة المشاريع مثل : (Web API + Class Library + Console App)

```
MyApp.sln
|
├─ MyApp.API
├─ MyApp.Services
└─ MyApp.Tests
```