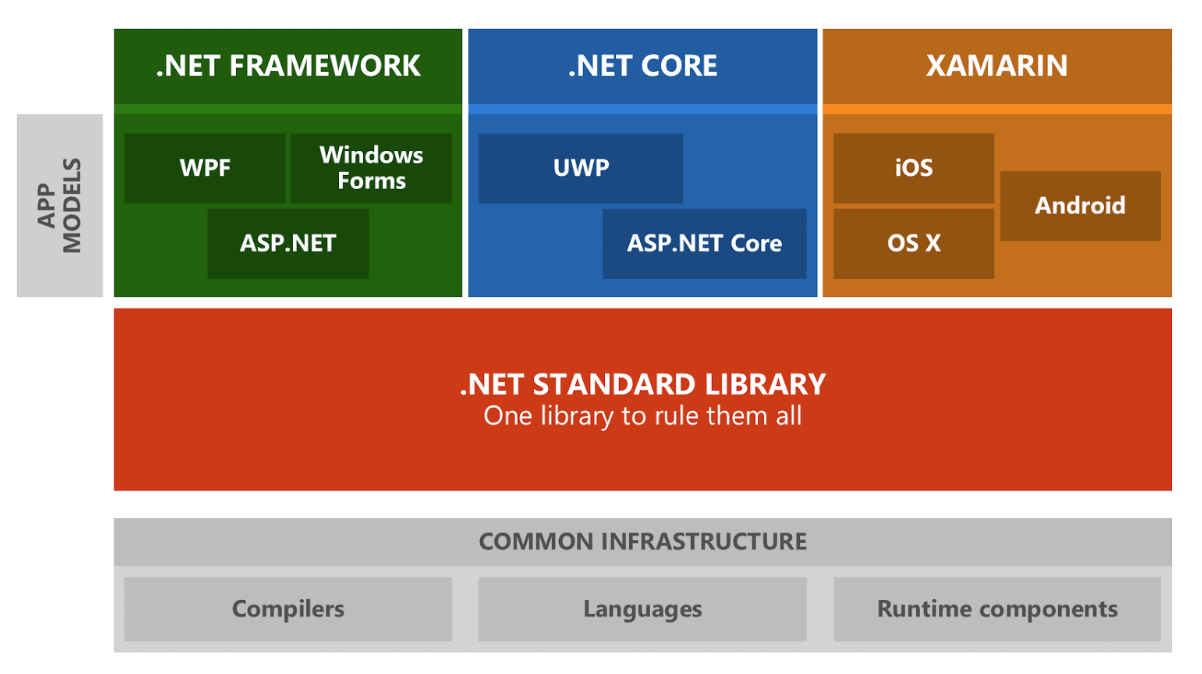
**Net CORE Vs ,Net**

**Name: Ahmad Sabry Aly**

**ID:2016170200\**

**Microsoft stack**

This is the architecture of Microsoft’s technology stack, which consists of 3 development implementations (.NET Framework, .NET Core and Mono) along with an API specification common to all of them, which is called .NET Standard.



**Differences between .NET Framework and .NET Core**

.NET Framework is the traditional .NET platform. It is Monolithic (you need to install all its base or you can’t use it) and it allows the creation of console, desktop, web and mobile applications. These applications run only on Windows.

.NET Core is the new generation of .NET technologies. Unlike its traditional version, .NET Core is based and developed with the following goals in mind:

* Lighter and more modular: This allows our application to provide exclusively what we need, without the need of installing the full .NET suite.
* Multi-platform: The applications that we create will work on Windows, Linux, and MacOS.
* High performance: .NET Core has a higher performance than the traditional version, which is very important for Cloud environments, where this translates into a lot of money saving over time.

**Migrating .NET applications to the new .NET Core. Can it be done?**

Let’s take a look at the different types of application and how difficult it is to migrate in each case. There are three main groups:

**Web applications:**

* Migration is simple for Backend and Frontend applications in ASP.NET MVC.
* It is not feasible with Frontend applications that use Web Forms. In that case, you have to do it from scratch.

**Applications designed for Windows 8/10:**

* If the application uses the universal platform, it is not necessary to migrate.
* In the case of applications based on Silverlight, the migration is very simple.
* If they are desktop applications based on WPF then the migration is complicated.
* It is not feasible for applications with Web Forms.

**Console application:**

* This will depend a lot on the application itself and the libraries that are being used. They can be of medium to hard difficulty.

There is also a set of important questions that might help determine the degree to which a project can be migrated to .NET Core:

* Do the applications use C# as their main programming language?
* Do they use the `.csproj` project file format? What libraries are included on that/those files? Are those libraries available for .NET Core?
* Are there any external services being used? Which?
* Does the project use Membership Provider services for user management?

**When should I use .NET Core?**

We can define 6 typical scenarios in which .NET Core is a good alternative:

1. When we need a multiplatform application, that works in several Operative Systems.
2. When we need to create applications based on a microservice architecture.
3. When we need scalable systems with high-performance needs.
4. When we need to work on developments for command line applications that work on Windows, Mac and Linux.
5. When we need to use several versions of .NET “side by side” in the same project.
6. When we need Universal Windows 10 applications (UWP), which use .NET Core on the back.

**Sustainability over time**

Today, .NET Core has a huge community, driven by the use of open source technologies and with an awesome GitHub community. At this moment (December 2018) .NET Core is available on versions 2.1.6 and 2.2 (announced very recently). At this point, it is a stable technology equipped with functionality that allows us to create high-capacity and multi-platform applications.