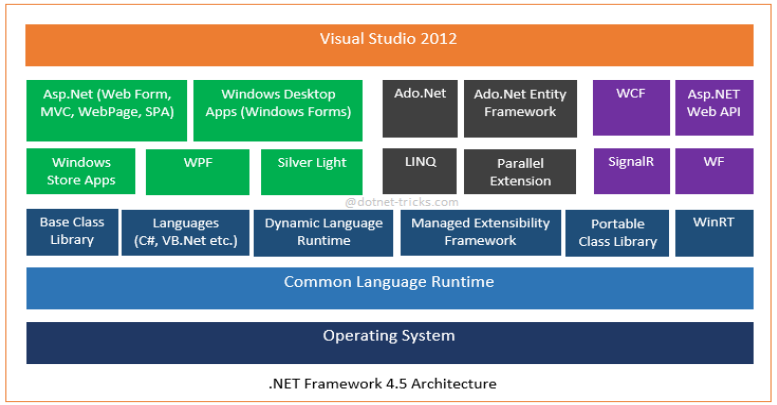
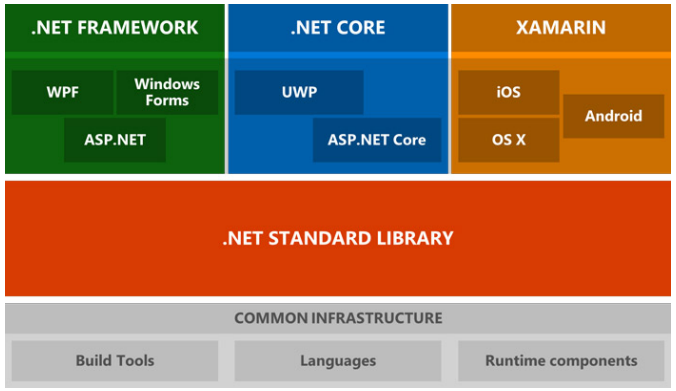


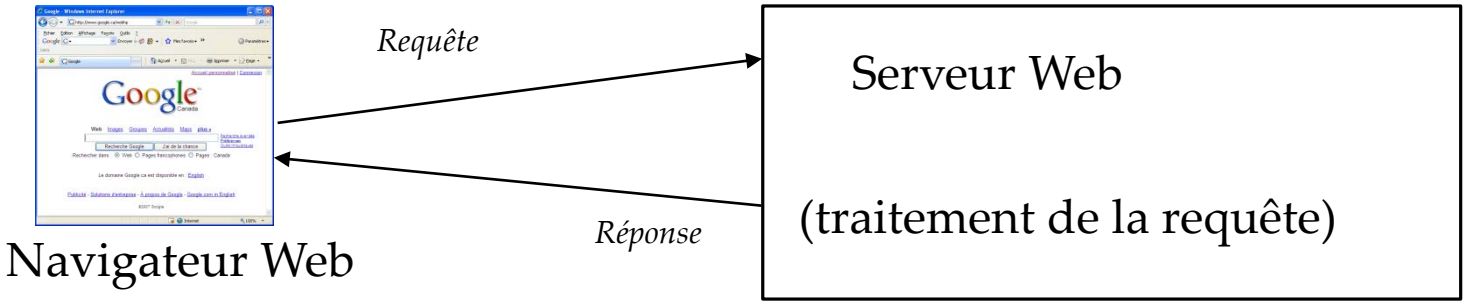


Introduction à la plateforme .NET pour le développement des applications Web modulaires.

MOHAMED AIROUCHE

AUTOMNE 2020

Approches *client-serveur*



Scénario de base d'une application Web

Sur Internet, deux ordinateurs communiquent et échangent des informations. Le scénario de base comprend un serveur (disposant de l'information) et un client (demandant de l'information).

- Première approche : *Requête/Réponse*
- Deuxième approche : *Traitements serveur*

Approches *client-serveur*

Première approche : ***Requête/Réponse***

Dans cette première approche, le client adresse une requête au serveur qui se charge de répondre au client en lui envoyant l'information demandée. Cette approche est simple, mais a pour principal désavantage d'être statique.

L'échange entre le client et le serveur s'effectue de la façon suivante:

1. Le client (navigateur Web) se connecte à un serveur à l'aide d'une URL. Le client effectue une demande sur une page spécifique (ex. index.htm).

2. Le serveur expédie le document demandé au navigateur client. Le client reçoit le document et le navigateur l'affiche.

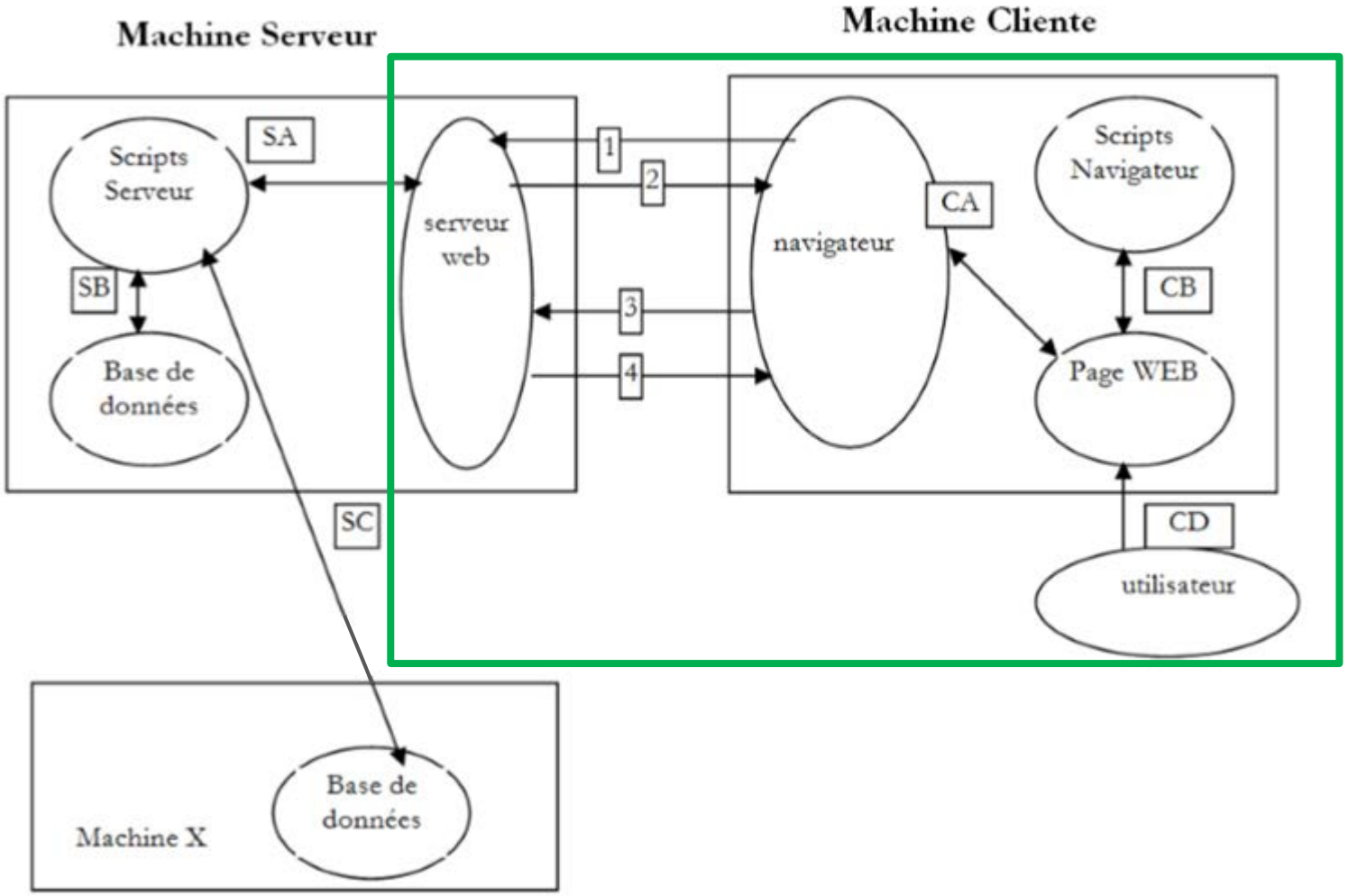
Deuxième approche : ***Traitements serveur***

Dans cette deuxième approche, le serveur effectue un traitement à son niveau avant de retourner la réponse au client. L'échange entre le client et le serveur s'effectue de la façon suivante :

1. Le client (navigateur Web) se connecte à un serveur à l'aide d'une URL. Le client effectue une demande sur une page spécifique (ex. : *index.php*, *index.asp*).

2. **Le serveur analyse la requête et traite son code.** Le client reçoit le document et le navigateur l'affiche.

Approches *client-serveur*



Introduction à ASP.NET MVC par l'exemple, Serge Tahe, novembre 2013. https://tahe.developpez.com/

Approches *client-serveur*

L'approche : *Traitements serveur*

1. Le navigateur demande une URL pour la 1ère fois (http://machine/url). Aucun paramètre n'est passé.

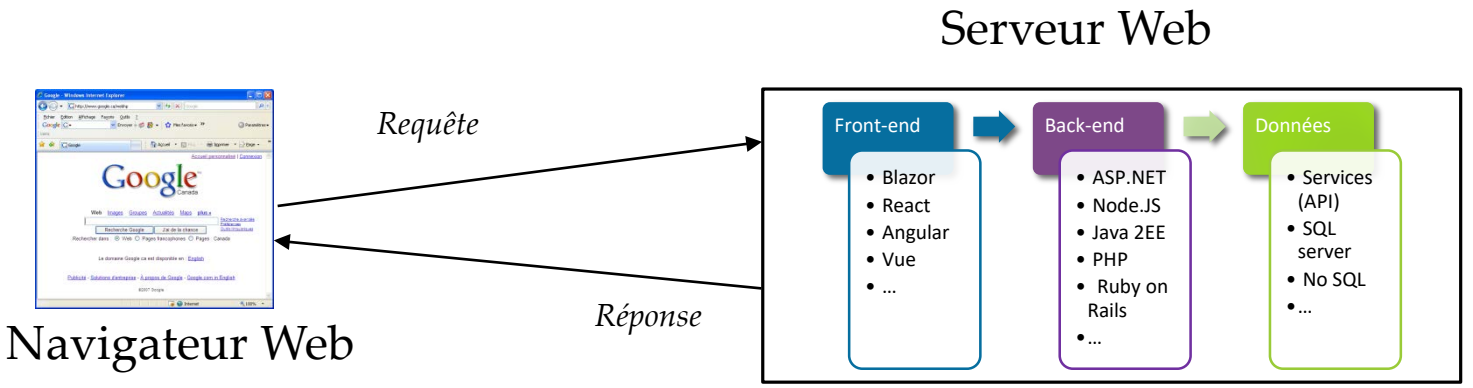
2. Le serveur Web lui envoie la page Web de cette URL. Elle peut être statique ou bien dynamiquement générée par un script serveur (SA) qui a pu utiliser le contenu de bases de données (SB, SC). Ici, le script détectera que l'URL a été demandée sans passage de paramètres et génèrera la page Web initiale.

   Le navigateur reçoit la page et l'affiche (CA). Des scripts côté navigateur (CB) ont pu modifier la page initiale envoyée par le serveur. Ensuite par des interactions entre l'utilisateur (CD) et les scripts (CB) la page Web va être modifiée. Les formulaires vont notamment être remplis.

3. L'utilisateur valide les données du formulaire qui doivent alors être envoyées au serveur Web. Le navigateur redemande l'URL initiale ou une autre selon les cas et transmet en même temps au serveur les valeurs du formulaire. Il peut utiliser pour ce faire deux méthodes appelées GET et POST. À réception de la demande du client, le serveur déclenche le script (SA) associé à l'URL demandée, script qui va détecter les paramètres et les traiter.

4. Le serveur délivre la page Web construite par programme (SA, SB, SC). Cette étape est identique à l'étape 2 précédente. Les échanges se font désormais selon les étapes 2 et 3.

Architecture d'une application Web



Une application Web moderne peut être composée principalement par les trois parties suivantes :

- Partie Front-end : contient la logique permettant de créer l'interface utilisateur et le script exécuté du coté client.
- Partie Back-end : contient la logique et le code exécuté du coté serveur comme la logique des couches métier et accès aux données.
- Partie des données : contient généralement les serveurs des données sous forme SQL ou non SQL.

Architecture d'une application Web

Les approches pour développer une application Web moderne

Vous avez le choix entre les approches suivantes pour créer des applications Web :

- Les applications Web traditionnelles qui effectuent la plupart de la logique d'application sur le serveur (pas de partie frontend).
- Les applications monopages qui effectuent la plupart de la logique d'interface utilisateur dans un navigateur Web en communiquant avec le serveur Web principalement à l'aide des API Web (Blazor+ Web API).
- Une approche hybride est également possible, la plus simple étant d'héberger une ou plusieurs sous-applications de type monopage au sein d'une application Web classique plus grande.

https://docs.microsoft.com/fr-ca/dotnet/architecture/modern-web-apps-azure/modern-web-applications-characteristics

Architecture d'une application Web

une application Web peut-être soit :

- **Une application monolithique :**

L'architecture de l'application peut se réduire à un seul projet avec une seule couche(non modulaire).

- **Une application utilisant l'architecture en couches:**

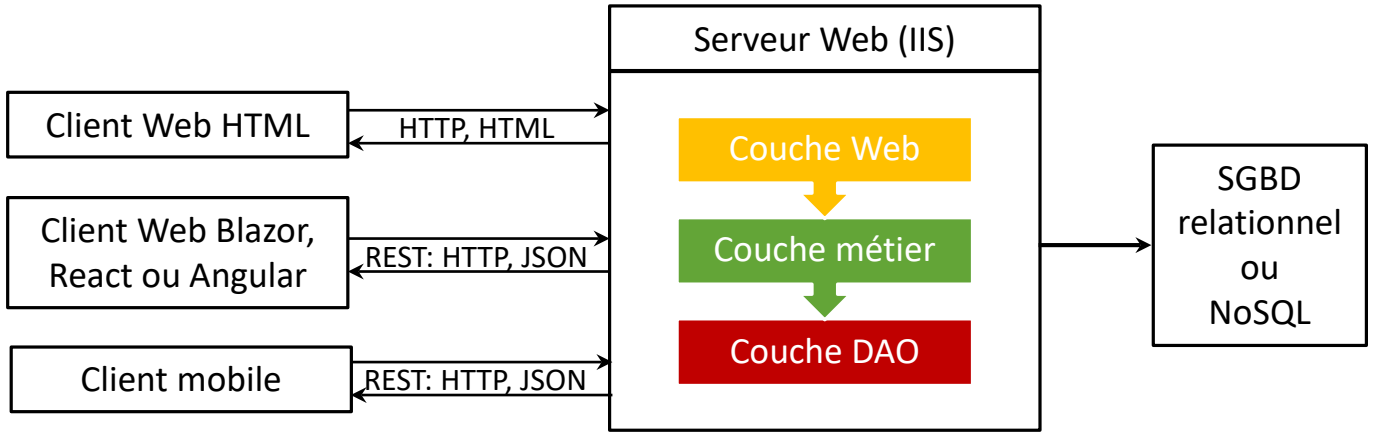
Quand une application devient complexe, un moyen de gérer cette complexité est de scinder l'application selon ses responsabilités ou préoccupations. Cela fait suite au principe de séparation des préoccupations et peut aider à maintenir une base de code croissante organisée afin que les développeurs puissent facilement trouver où certaines fonctionnalités sont mises en œuvre. L'architecture en couches offre d'autres avantages que la simple organisation du code.

https://docs.microsoft.com/fr-ca/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

Architecture d'une application Web

Une application Web utilisant une architecture en couches est souvent composée de trois couches :

- La couche Web
- La couche métier
- La couche d'accès aux bases de données



https://docs.microsoft.com/fr-ca/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

La programmation Web

Côté client

Une seule chose à retenir : on devrait s'assurer que la majorité du traitement à effectuer se fasse côté client pour libérer le plus possible le serveur (ex. : dynamisme des pages – DHTML, validation des informations, etc.).

Les langages utilisés sont JavaScript et VB Script et récemment C#.

Les Frameworks utilisés:

Bootstrap, JQuery, Blazor, React, Angular, …

Côté serveur

La programmation serveur se situe à deux niveaux :

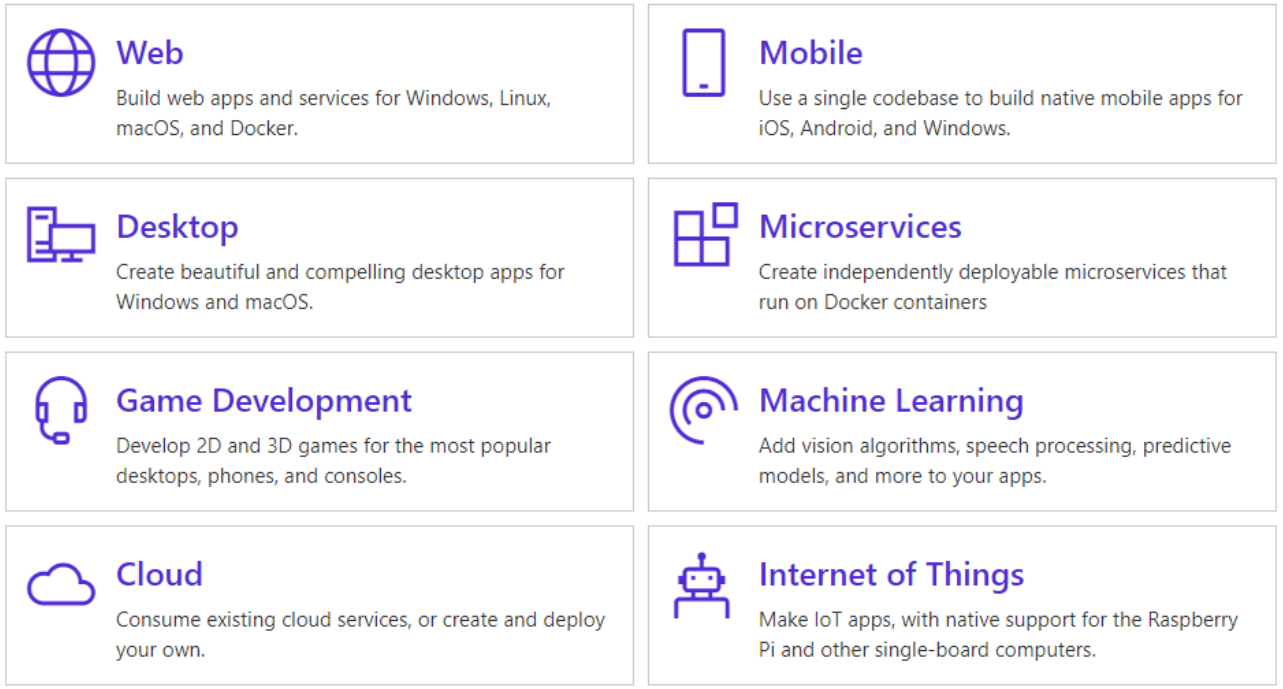
✓Contenu dynamique des pages Web (génération du code XHTML en fonction de certains paramètres).

✓Accès SGBD.

Les principaux langages utilisés sont **Java – Servlets & JSP, PHP, ASP et ASP.NET, Node JS, Python, Ruby on rails.**

La plateforme de développement .NET

- .NET est une plateforme de développement open source permettant de créer différent types d'applications.
- Plateforme de développement = langages + librairies



https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

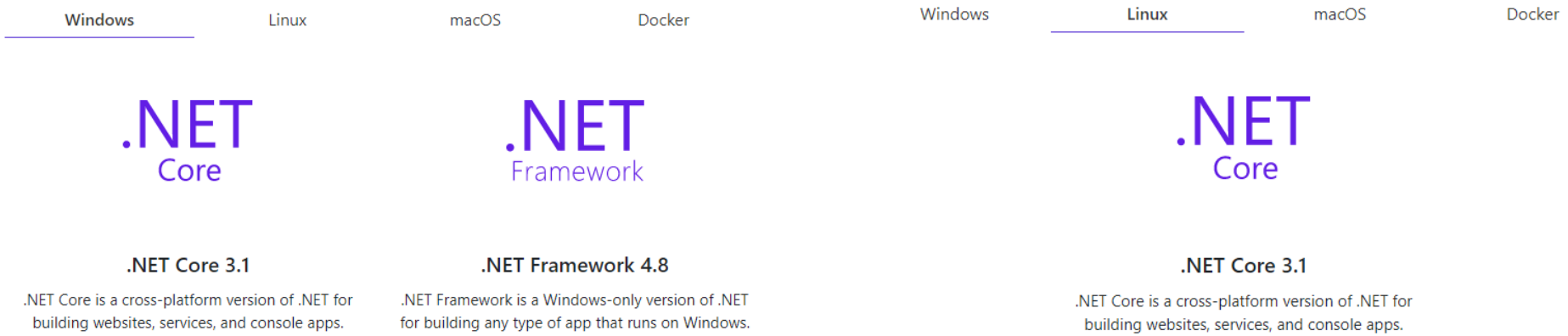
https://dotnet.microsoft.com/

La plateforme de développement .NET

- .NET Framework est l'implémentation originale de .NET. Il prend en charge l'exécution de sites Web, de services, d'applications de bureau et plus encore sur Windows.

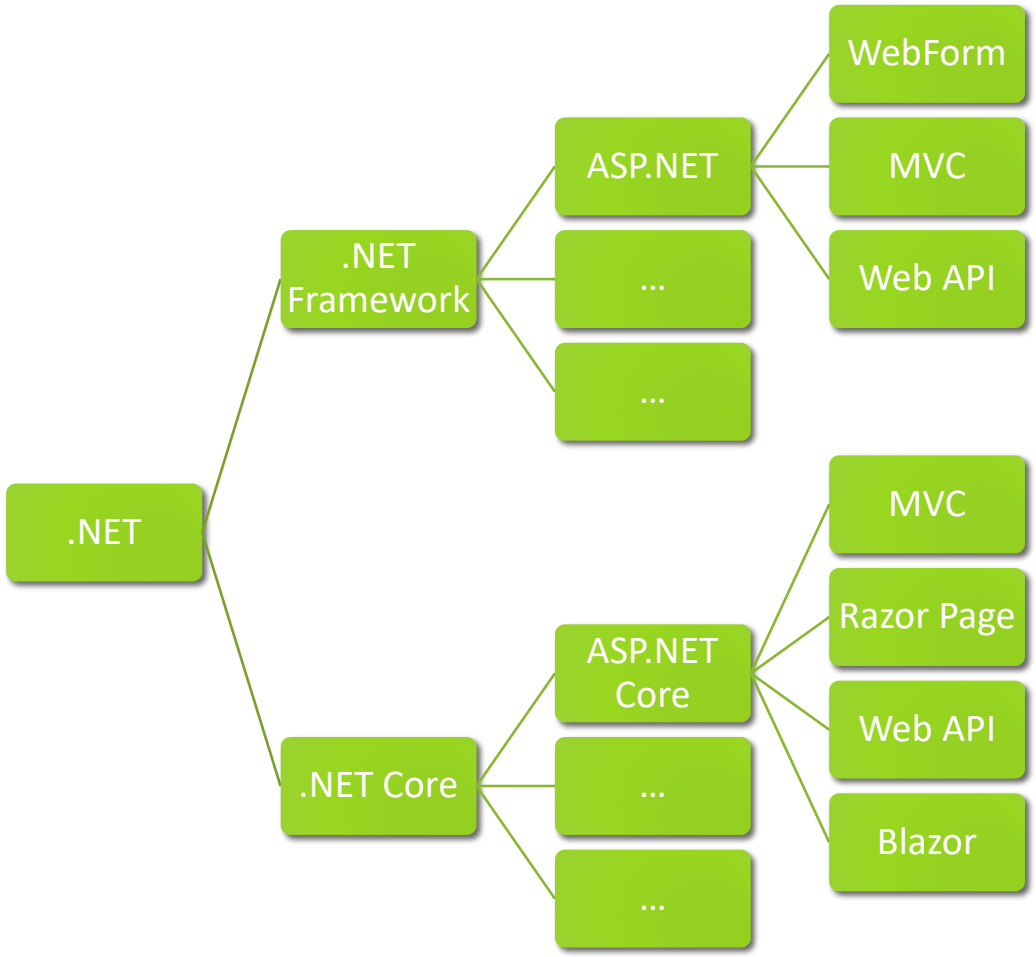
  https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

- .NET Core est une version multiplateforme de .NET pour la création de sites Web, de services et d'applications de console. .NET Core est une réécriture complète du .NET Framework, indépendant de la plateforme et des langages et open source. Afin d'être en mesure de s'exécuter sur des environnements autres que Windows tels que Linux et MacOS.



https://dotnet.microsoft.com/download

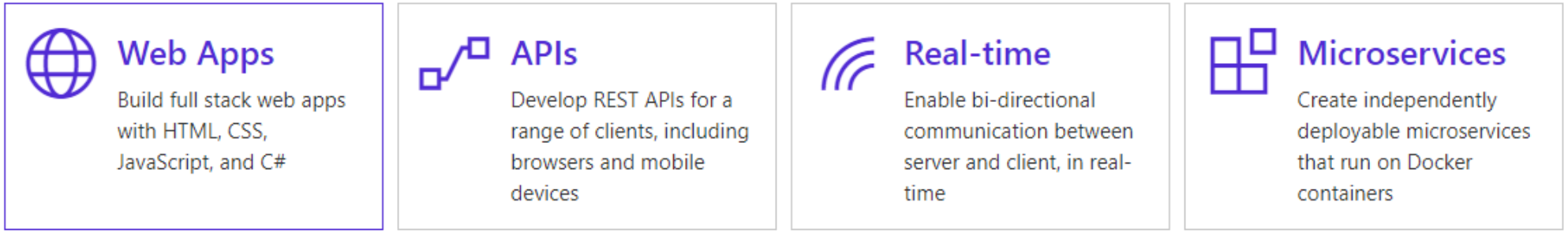
La plateforme de développement .NET



ASP.NET

- ASP.NET étend la plateforme de développement .NET avec des outils et des bibliothèques spécifiquement pour la création d'applications Web.
- ASP.NET est une plateforme(environnement) de développement de Microsoft pour réaliser des applications Web.
- Il est important de noter qu'ASP.NET n'est pas un langage de programmation, mais plutôt **une technologie pour concevoir des applications destinées au Web**.

https://www.youtube.com/watch?v=lE8NdaX97m0&list=PLdo4fOcmZ0oW8nviYduHq7bmKode-p8Wy&index=1



https://dotnet.microsoft.com/apps/aspnet

ASP.NET

- **ASP.NET 4.x :** ASP.NET 4.x (ASP.NET) est un Framework qui offre les services nécessaires pour créer sur Windows des applications Web, basées côté serveur. (https://docs.microsoft.com/fr-fr/aspnet/overview)
  - WebForms et MVC sont deux logiques différentes de développement utilisées en ASP.NET.
  - ASP.NET WebForms a été créé en 2002 et aujourd'hui elle est à sa version 5.
  - ASP.NET MVC est plus récent et a fait son apparition en 2008.

- **ASP.NET Core** est une nouvelle conception de ASP.NET 4.x, y compris des modifications architecturales qui se traduisent par un Framework plus léger et plus modulaire. ASP.NET Core est une infrastructure multiplateforme, à hautes performances et Open source pour la création d'applications modernes, basées sur le Cloud et Internet des objets connectés. (https://docs.microsoft.com/fr-ca/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1)
  - MVC, Razor Pages, Web API sont différentes logiques de développement utilisées du côté serveur en ASP.NET Core.
  - Blazor vous permet d'utiliser C# dans le navigateur en même temps que JavaScript. Il permet de partager les logiques d'applications côté serveur et côté client écrits avec .NET.

Historique de l'ASP.NET

1996 - Les Active Server Pages (ASP classique)

2002 - .NET 1.0 (Arrivée de l'ASP.NET : formulaires Web)

2008 - .NET 3.5 (Première version de MVC - Framework ASP.NET MVC)

2010 - .NET 4 (VS 2010, MVC 2.0)

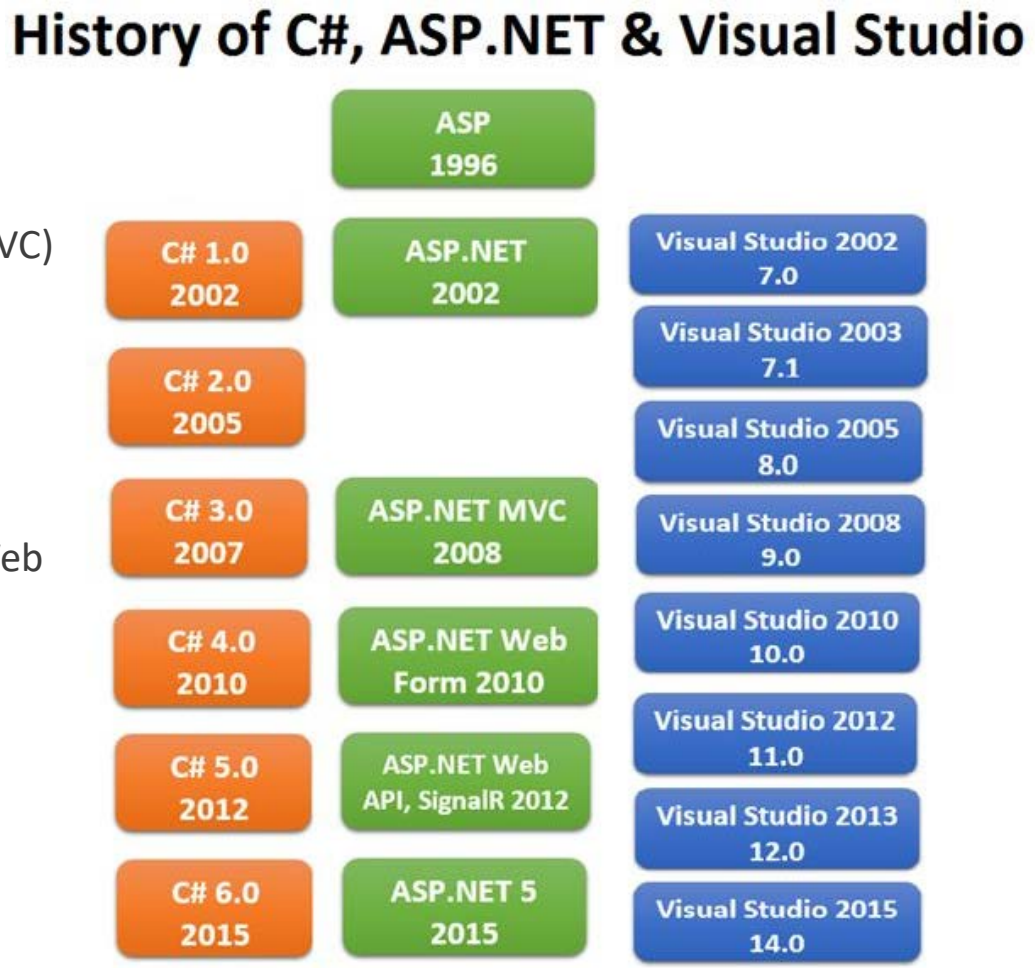
2011 - .NET 4 (MVC 3.0, Razor)

2012 - .NET 4.5 (MVC 4.0 , Première version de WebAPI - service Web REST, VS 2012)

2013 - .NET 4.5 (Visual Studio 2013, MVC 5.0)

2014 - .NET 4.5 (MVC 5.2)

2016 - .NET Core (MVC 6 → MVC core 1.0)



Historique de l'ASP.NET Core

ASP.NET Core est la version open source d'**ASP.NET** de Microsoft, il a été publié pour la première fois en 2016.

2016 : .Net Core 1.0 (ASP.NET Core MVC)

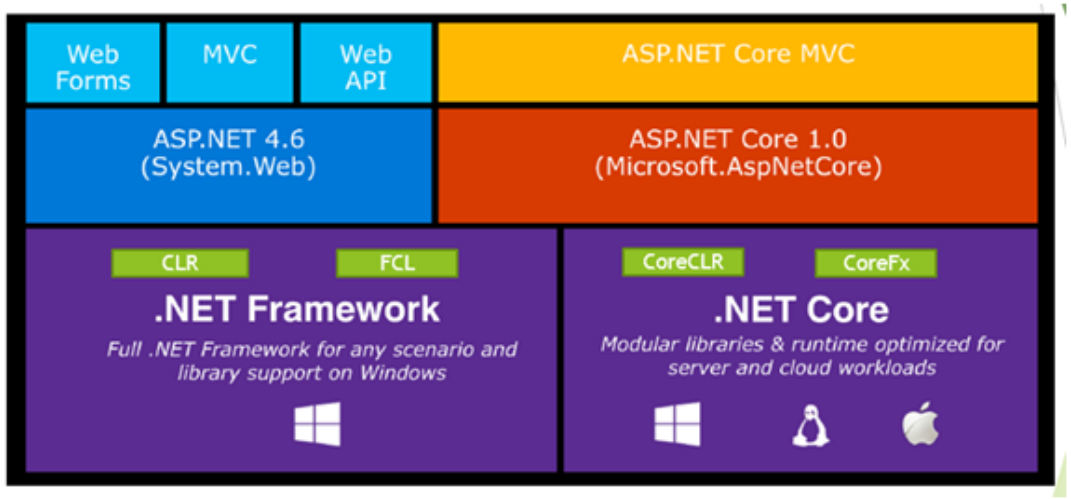
2017 : .Net Core 2.0 (MVC+Razor Pages + API)

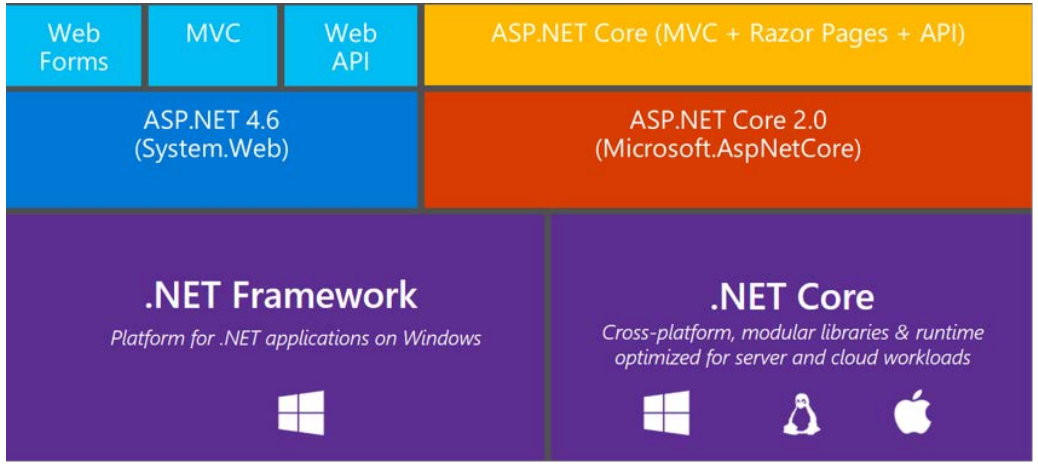
2018 : .Net Core 2.1 - LTS

2019 : .Net Core 3.0 (+Blazor)

Novembre 2019 : .Net Core 3.1 - LTS

Novembre 2020 : .Net 5.0





Historique de l'ASP.NET Core

2016 : .Net Core 1.0 (ASP.NET Core MVC)

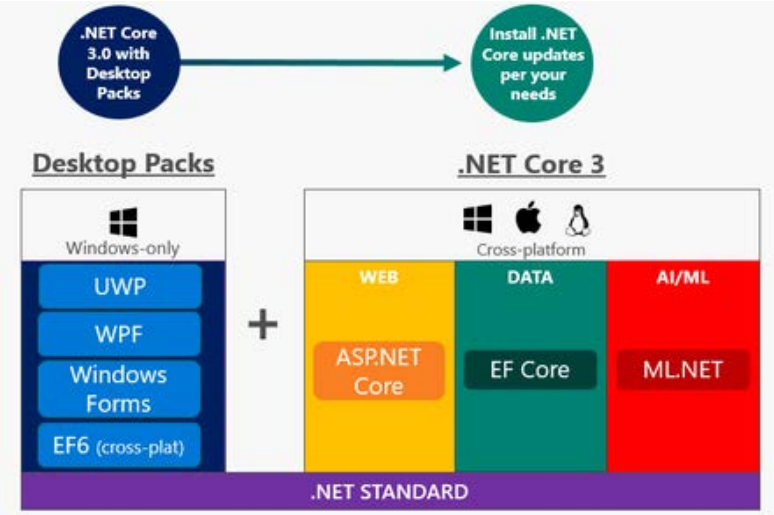
2017 : .Net Core 2.0 (MVC+Razor Pages + API)

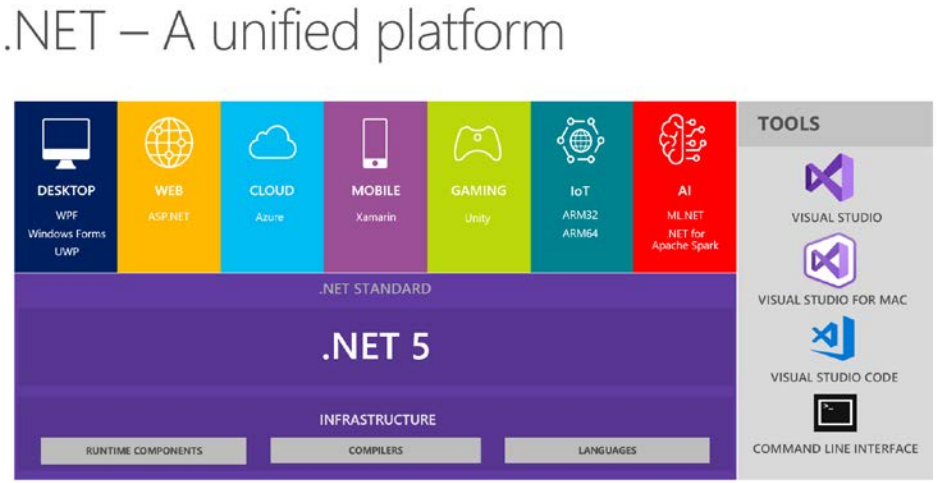
2018 : .Net Core 2.1 LTS

2019 : .Net Core 3.0 (+Blazor)

Novembre 2019 : .Net Core 3.1 LTS

Novembre 2020 : .Net 5.0





ASP.NET Core 3.1 est la dernière version de cette technologie open-source et multiplateformes proposées par Microsoft, qui est soutenue par une importante communauté.

Historique de l'ASP.NET Core

2016 : .Net Core 1.0 (ASP.NET Core MVC)

2017 : .Net Core 2.0 (MVC+Razor Pages + API)

2018 : .Net Core 2.1 LTS

2019 : .Net Core 3.0 (+Blazor)

Novembre 2019 : .Net Core 3.1 LTS

Novembre 2020 : .Net 5.0

ASP.NET Core 3.1 est la dernière version de cette technologie open-source et multiplateformes proposées par Microsoft, qui est soutenue par une importante communauté.

- .NET Core 3.0 release in September
- .NET Core 3.1 = Long Term Support (LTS)
- .NET 5.0 release in November 2020
- Major releases every year, LTS for even numbered releases
- Predictable schedule, minor releases if needed

Les parties à étudier dans le cours

Dans la suite de ce cours, nous nous intéresserons à:

- La logique de développement MVC avec ASP.NET (architecture MVC, routage, validation et liaison de données).
- La logique d'accès aux bases de données (ADO.NET, DbContext, LINQ, l'ORM Entity Framework).
- Les logiques de développement avec ASP.NET Core (modularité, injection de dépendance, MVC, Pages Razor, Web API).
- Utilisation d'Identity Framework pour la sécurité de l'application (authentification et autorisation).
- La logique de développement de service Web (API) du côté serveur et Blazor du côté client en utilisant juste la technologie .NET.

https://dotnet.microsoft.com/learn/aspnet/architecture

https://dotnet.microsoft.com/learn/aspnet