**creare nuova directory e progetto ASP.NET per il web da riga di comando**

* Posizionarsi sulla cartella in cui vogliamo creare la cartella progetto

C:\Users\Utente>cd Documents

C:\Users\Utente\Documents>**cd** visual progetti

C:\Users\Utente\Documents\visual progetti>cd codice

* Di seguito creiamo la cartella del progetto

C:\Users\Utente\Documents\visual progetti\codice>**mkdir** MyCourse

C:\Users\Utente\Documents\visual progetti\codice>cd MyCourse

C:\Users\Utente\Documents\visual progetti\codice\MyCourse>

* Di seguito creiamo il progetto vuoto per ASP.NET

C:\Users\Utente\Documents\visual progetti\codice\MyCourse>**dotnet new web**

Creazione del modello "ASP.NET Core Empty" completata.

Elaborazione post-creazione delle azioni in corso...

Esecuzione di 'dotnet restore ' in C:\Users\Utente\Documents\visual progetti\codice\MyCourse\MyCourse.csproj...

Individuazione dei progetti da ripristinare...

C:\Users\Utente\Documents\visual progetti\codice\MyCourse\MyCourse.csproj ripristinato (in 69 ms).

Il ripristino è riuscito.

Il comando ha creato alcuni file che posso elencare con **dir**

C:\Users\Utente\Documents\visual progetti\codice\MyCourse>dir

Il volume nell'unità C non ha etichetta.

Numero di serie del volume: BE03-C469

Directory di C:\Users\Utente\Documents\visual progetti\codice\MyCourse

21/04/2022 08:38 <DIR> .

21/04/2022 08:38 <DIR> ..

21/04/2022 08:38 127 appsettings.Development.json

21/04/2022 08:38 151 appsettings.json

21/04/2022 08:38 219 MyCourse.csproj

21/04/2022 08:38 <DIR> obj

21/04/2022 08:38 135 Program.cs

21/04/2022 08:38 <DIR> Properties

4 File 632 byte

4 Directory 152.891.338.752 byte disponibili

* Di seguito lanciamo l’applicazine con il comando **dotnet run**

C:\Users\Utente\Documents\visual progetti\codice\MyCourse>**dotnet run**

Compilazione...

info: Microsoft.Hosting.Lifetime[14]

Now listening on: **https://localhost:7130**

info: Microsoft.Hosting.Lifetime[14]

Now listening on: **http://localhost:5044**

info: Microsoft.Hosting.Lifetime[0]

Application started. Press Ctrl+C to shut down.

info: Microsoft.Hosting.Lifetime[0]

Hosting environment: Development

info: Microsoft.Hosting.Lifetime[0]

Content root path: C:\Users\Utente\Documents\visual progetti\codice\MyCourse\

vediamo come la nostra app si va a posizionare su due indirizzi localhost, 7130 e 5044

Per accedere alle funzioni di .NET SDK si usa il comando da riga di comando

**dotnet** seguito da *un qualcosa*.

Questo qualcosa è un comando come run o new ad esempio

Per vederli tutti si usa

**dotnet --help**

Per avere l’help relativo al comando che ci interessa, es. **new**

**dotnet new --help**

e per avere la lista dei template (modelli) disponibili, possiamo averla con

**dotnet new --list**

Nome modello Nome breve Lingua Tag

-------------------------------------------- ------------------- ---------- --------------------------

App console console [C#],F#,VB Common/Console

App Windows Forms winforms [C#],VB Common/WinForms

Applicazione WPF wpf [C#],VB Common/WPF

ASP.NET Core Empty web [C#],F# Web/Empty

ASP.NET Core gRPC Service grpc [C#] Web/gRPC

ASP.NET Core Web API webapi [C#],F# Web/WebAPI

ASP.NET Core Web App webapp,razor [C#] Web/MVC/Razor Pages

ASP.NET Core Web App (Model-View-Controller) mvc [C#],F# Web/MVC

ASP.NET Core with Angular angular [C#] Web/MVC/SPA

ASP.NET Core with React.js react [C#] Web/MVC/SPA

Blazor Server App blazorserver [C#] Web/Blazor

Blazor WebAssembly App blazorwasm [C#] Web/Blazor/WebAssembly/PWA

Configurazione NuGet nugetconfig Config

Configurazione Web webconfig Config

File di soluzione sln Solution

File EditorConfig editorconfig Config

file gitignore dotnet gitignore Config

file global.json globaljson Config

File manifesto dello strumento locale dotnet tool-manifest Config

Libreria di classi classlib [C#],F#,VB Common/Library

Libreria di classi Windows Form winformslib [C#],VB Common/WinForms

Libreria di classi WPF wpflib [C#],VB Common/WPF

Libreria di controlli personalizzati WPF wpfcustomcontrollib [C#],VB Common/WPF

Libreria di controlli utente WPF wpfusercontrollib [C#],VB Common/WPF

Libreria di controlli Windows Form winformscontrollib [C#],VB Common/WinForms

MSTest Test Project mstest [C#],F#,VB Test/MSTest

MVC ViewImports viewimports [C#] Web/ASP.NET

MVC ViewStart viewstart [C#] Web/ASP.NET

NUnit 3 Test Item nunit-test [C#],F#,VB Test/NUnit

NUnit 3 Test Project nunit [C#],F#,VB Test/NUnit

Protocol Buffer File proto Web/gRPC

Razor Class Library razorclasslib [C#] Web/Razor/Library

Razor Component razorcomponent [C#] Web/ASP.NET

Razor Page page [C#] Web/ASP.NET

Worker Service worker [C#],F# Common/Worker/Web

xUnit Test Project xunit [C#],F#,VB Test/xUnit

con **dotnet new web** abbiamo usato il Template **web** per settare la nostra applicazione

mettendo **dotnet new web --help** possiamo vedere i parametri del Template web

C:\Users\Utente\Documents\visual progetti\codice\MyCourse>dotnet new web --help

ASP.NET Core Empty (C#)

Autore: Microsoft

Descrizione: An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

Opzioni:

--exclude-launch-settings Whether to exclude launchSettings.json from the generated template.

bool - Optional

Impostazione predefinita: false

--no-restore If specified, skips the automatic restore of the project on create.

bool - Optional

Impostazione predefinita: false

--no-https Whether to turn off HTTPS. This option only applies if Individual, IndividualB2C,SingleOrg, or MultiOrg aren't used for --auth.

bool - Optional

Impostazione predefinita: false

se voglio cambiare la versione di dotnet da usare posso fare con

**dotnet new --list**

e vedrò che c'e un template che si chiama **global.jason file** che punta alla versione che sto usando

posso creare una nuova istanza di global.json (vedi elenco con file evidenziato )

**dotnet new globaljson**

se poi faccio

**type glogal.json**

ottengo il contenuto del file

C:\Users\Utente\Documents\visual progetti\codice\MyCourse>type global.json

{

"sdk": {

"version": "6.0.202"

}

}

se voglio sapere tutte le versioni SDK installate posso fare

**dotnet --list-sdks**

per scegliere un'altra versione basta modificare il file con

**notepad global.json**

lo apro con notepad e cambio la versione di riferimento. Salvo. Se poi rifaccio

dotnet --version

ottengo la versione di SDK nuova che ho scelto tra quelle che avevo scaricato

se voglio tornare alla versione precedente devo solo eliminare il file global.json e torno alla versione predefinita

del global.json

Aggiungere un package da nuget

dotnet add package Microsoft.Data.Sqlite.Core --version 6.0.4

e ci dice anche se è compatibile.

C:\Users\Utente\Documents\visual progetti\codice\MyCourse>dotnet add package Microsoft.Data.Sqlite.Core --version 6.0.4

Individuazione dei progetti da ripristinare...

Writing C:\Users\Utente\AppData\Local\Temp\tmp15C0.tmp

info : Aggiunta dell'oggetto PackageReference per il pacchetto 'Microsoft.Data.Sqlite.Core' nel progetto 'C:\Users\Utente\Documents\visual progetti\codice\MyCourse\MyCourse.csproj'.

info : Ripristino dei pacchetti per C:\Users\Utente\Documents\visual progetti\codice\MyCourse\MyCourse.csproj in corso...

info : GET https://api.nuget.org/v3-flatcontainer/microsoft.data.sqlite.core/index.json

info : OK https://api.nuget.org/v3-flatcontainer/microsoft.data.sqlite.core/index.json 523 ms

info : GET https://api.nuget.org/v3-flatcontainer/microsoft.data.sqlite.core/6.0.4/microsoft.data.sqlite.core.6.0.4.nupkg

info : OK https://api.nuget.org/v3-flatcontainer/microsoft.data.sqlite.core/6.0.4/microsoft.data.sqlite.core.6.0.4.nupkg 42 ms

info : GET https://api.nuget.org/v3-flatcontainer/sqlitepclraw.core/index.json

info : OK https://api.nuget.org/v3-flatcontainer/sqlitepclraw.core/index.json 514 ms

info : GET https://api.nuget.org/v3-flatcontainer/sqlitepclraw.core/2.0.6/sqlitepclraw.core.2.0.6.nupkg

info : OK https://api.nuget.org/v3-flatcontainer/sqlitepclraw.core/2.0.6/sqlitepclraw.core.2.0.6.nupkg 46 ms

info : GET https://api.nuget.org/v3-flatcontainer/system.memory/index.json

info : OK https://api.nuget.org/v3-flatcontainer/system.memory/index.json 520 ms

info : GET https://api.nuget.org/v3-flatcontainer/system.memory/4.5.3/system.memory.4.5.3.nupkg

info : OK https://api.nuget.org/v3-flatcontainer/system.memory/4.5.3/system.memory.4.5.3.nupkg 62 ms

info : SQLitePCLRaw.core 2.0.6 installato da https://api.nuget.org/v3/index.json con hash del contenuto Vh8n0dTvwXkCGur2WqQTITvk4BUO8i8h9ucSx3wwuaej3s2S6ZC0R7vqCTf9TfS/I4QkXO6g3W2YQIRFkOcijA==.

info : Microsoft.Data.Sqlite.Core 6.0.4 installato da https://api.nuget.org/v3/index.json con hash del contenuto 3TZX7R2aX1TX5m4A5Kj+SY633NJDeHDP6JiDRCwUnJGKC3IrHgnO8p+oT2hRZpN168qx4Ixe4T9C+xZdZc26gw==.

info : System.Memory 4.5.3 installato da https://api.nuget.org/v3/index.json con hash del contenuto 3oDzvc/zzetpTKWMShs1AADwZjQ/36HnsufHRPcOjyRAAMLDlu2iD33MBI2opxnezcVUtXyqDXXjoFMOU9c7SA==.

info : Il pacchetto 'Microsoft.Data.Sqlite.Core' è compatibile con tutti i framework specificati nel progetto 'C:\Users\Utente\Documents\visual progetti\codice\MyCourse\MyCourse.csproj'.

info : L'oggetto PackageReference per il pacchetto 'Microsoft.Data.Sqlite.Core' versione '6.0.4' è stato aggiunto al file 'C:\Users\Utente\Documents\visual progetti\codice\MyCourse\MyCourse.csproj'.

info : Scrittura del file di asset sul disco. Percorso: C:\Users\Utente\Documents\visual progetti\codice\MyCourse\obj\project.assets.json

log : C:\Users\Utente\Documents\visual progetti\codice\MyCourse\MyCourse.csproj ripristinato (in 2,9 sec).

**dotnet –-version** ho la versione di dotnet

**dotnet build** compila la applicazione ma non la lancia

**dotnet run** compila e lancia la applicazione

**dotnet add** serve per aggiungere funzioni, es. pacchetti (tipo per paypal, per la fatturazione elettronica ecc. (vedi sito https://www.nuget.org/), in questo caso usiamo **dotnet add package**

certificati SSL dal Web

l'applicazione si colloca su due indirizzi localhost

info: Microsoft.Hosting.Lifetime[14]

Now listening on: **https://localhost:7130**

info: Microsoft.Hosting.Lifetime[14]

Now listening on: **http://localhost:5044**

il primo è un indirizzo https perché Microsoft vorrebbe che accettassimo solo connessioni sicure

se entriamo in [**https://localhost:7130**](https://localhost:7130)il sistema ci dice che la connessione non è privata perché non c'è un certificato valido

dobbiamo dirgli di accettare quel certificato

con dotnet --help scopro che c'è un sottocomando dev-certs

se faccio dotnet dev-certs --help

Usage: dotnet dev-certs [options] [command]

Options:

-h|--help Show help information

Commands:

https

Use "dotnet dev-certs [command] --help" for more information about a command.

scopro che devo mettere dev-certs https --help per saperne di più

Options:

-ep|--export-path Full path to the exported certificate

-p|--password Password to use when exporting the certificate with the private key into a pfx file or to encrypt the Pem exported key

-np|--no-password Explicitly request that you don't use a password for the key when exporting a certificate to a PEM format

-c|--check Check for the existence of the certificate but do not perform any action

--clean Cleans all HTTPS development certificates from the machine.

-i|--import Imports the provided HTTPS development certificate into the machine. All other HTTPS developer certificates will be cleared out

--format Export the certificate in the given format. Valid values are Pfx and Pem. Pfx is the default.

-t|--trust Trust the certificate on the current platform. When combined with the --check option, validates that the certificate is trusted.

-v|--verbose Display more debug information.

-q|--quiet Display warnings and errors only.

-h|--help Show help information

con **--trust** posso dire di fidarmi del certificato sulla piattaforma corrente e Windows installerà il certificato e quindi lo riterrà attendibile.

dotnet dev-certs https --trust

**COMANDI BASE DI VISUAL STUDIO CODE**

CTRL + ò = apre in terminale sotto il progetto

fn+F1 = apre la command palette (posso fare tutto quello che posso fare cliccando su un menu

**per avviare una applicazione**

dotnet run

dotnet MyCourse.dll su server

oppure F5 così fa entrambe

**ricerche sulla command palette**

con CTRL+T appare #

cerca tutti i simboli, cioè ogni cosa che abbia un identificatore nel software, cioè un nome.

con CTRL+SHIFT+O appare @

cerco simboli nel file

con CTRL+P cerca i file dentro l'applicazione

con CTRL+G appare : e cerco la riga di codice

con ALT+F12 su una classe posso vedere i dettagli di quella classe e dove viene usata

con SHIFT+F12 vedo i riferimenti di una classe

con CTRL+SHIFT+B facciamo la Build del progetto

**Tipi di file in un progetto asp.net**

Quali file compongono un progetto e qual è la loro singola responsabilità

1) file con estensione .csproj

Serve a dare istruzioni di BUILD alla piattaforma .NET CORE su come deve essere compilata la nostra applicazione.

Qui vengono menzionati solo i file che non vanno compilati.

Vengono invece richiamati tutti i Pakage di riferimento.

2) file sorgenti .cs e .cshtml

.cs sono file C#

.cshtml sono file che servono per le interfacce View Razor – codice misto c# e html

**Cosa accade quando si compila**

MSBuild raccoglie i file .csproj, .cs e .cshtml, li organizza e li passa a ROSLYN che è il vero compilatore

ROSLYN butta fuori dei file assembly con estensione .dll che ha un codice detto *intermediate language (IL)*

Quando lanciamo la BUILD vengono prodotte due cartelle

bin

obj

obj: directory temporanea in cui il compilatore tiene dei semilavorati che gli serviranno per poi realizzare il vero output, che invece sta nella directory bin.

In bin c’è tutto l’output, compreso

* il file .dll che contiene il codice C# compilato.
* Il file .pdb che contiene una mappatura per collegare il codice C# e il codice del .dll

(È utile per il debugger perché così sa dove va ad interrompersi anche l’assembly relativo.)

* Il file deps.json che riepiloga tutte le dipendenze da pacchetti della nostra applicazione.
* Il file runtimeconfig.json contiene riferimenti alla specifica versione di .NET Core che deve essere usata.

bin e obj possono essere sempre eliminate perché vengono ricreate ogni volta che viene rilanciata la build.

Quando compiliamo il .dll l’assembly viene passato al sistema RyuJIT che è un compilatore JUST in TIME che converte in linguaggio macchina.

**Cosa succede quando si avvia l’applicazione**

Entra in gioco la classe Program che permette di prendere richieste web dagli utenti.

È definita in un file dal nome program.cs (convenzione: creare una classe all’interno di un file che porta lo stesso nome).

Program contiene il punto di ingresso della applicazione

public static void Main (string [ ], args)

in questo caso questa espressione riguarda le applicazioni di tipo CONSOLE, e in ASP.NET si tratta di applicazioni di un tipo particolare di console che per prima cosa fa

CreateWebHostBuilder(args) .Build() .Run();

In pratica Crea un WEB HOST , cioè un componente in grado di ricevere le chiamate degli utenti quando scrivono nella ricerca il loro <http://sito>.com ecc.. In esso esiste, integrato, un web server chiamato **Kestrel.**

Una applicazione ASP.NET Core contiene già tutto quello che gli serve per funzionare al suo interno.

Kestrel riceve le richiesta HTTP degli utenti e poi le converte in un oggetto chiamato

**HTTP context,** che è strutturato per essere passato al codice dell’applicazione che così può decidere come rispondere all’utente.

Con questo HTTP context possiamo scrivere all’utente attraverso l’operatore Response

In questo WEB HOST esistono anche:

un operatore di Logging che serve per dare dei messaggi diagnostici all’interno della console

un operatore di Configurazione che dice qual è la configurazione usata.

**Parametri di avvio**

Una applicazione asp.net viene avviata su due endpoint

sono comandati dal file launcsettings.json in cui si può modificare o rimuovere questi endpoint

"profiles": {

    "MyCourse": {

      "commandName": "Project",

      "dotnetRunMessages": true,

      "launchBrowser": true,

      "applicationUrl": "https://localhost:7041;http://localhost:5263",

      "environmentVariables": {

        "ASPNETCORE\_ENVIRONMENT": "Development"

possiamo eliminare l'endpoint http perché quello https è più che sufficiente.

oppure possiamo cambiare la porta 5263 se è già occupata

"applicationUrl": "https://localhost:7041;http://localhost:5264",

oppure

"applicationUrl": <https://localhost:7041;http://localhost:5264;http://192.168.1.126>:5003,

per condividere la visualizzazione host con un indirizzo fisico di un collega

<http://192.168.1.126> tramite la porta 5003 che non è occupata

**Middleware**

codice json organizzato in componenti che si trovano l'uno di seguito all'altro, ognuno gestisce parti della risposta al cliente. Sono dei componenti modulari che possiamo usare sia da microsoft che produrre noi stessi.

Servono in base all'ambiente e traducono il nostro codice in risposte web

I **middleware** sono componenti che offrono una propri funzionalità specifica e sono chiamati a gestire le richieste HTTP in ingresso. Sono **disposti l'uno dopo l'altro in una serie ordinata** che è anche chiamata "pipeline". Ogni middleware ha facoltà di:

* **Interagire con l'oggetto**HttpContext che rappresenta il contesto di esecuzione corrente. Questo oggetto può essere usato dal middleware per leggere le informazioni passate con la richiesta, per aggiungervi informazioni arbitrarie o riscriverla del tutto;
* Quando ha terminato il suo lavoro, il middleware può **cedere la gestione della richiesta al middleware successivo**, oppure impedire che questo avvenga. Quindi, non è detto che ogni richiesta venga gestita da tutti i middleware presenti nella serie;
* **Produrre una risposta** per il client o riscrivere o integrare la risposta prodotta in precedenza da un altro middleware.

L'**ordine in cui usiamo i middleware è importante**: il primo middleware della serie sarà il primo a gestire la richiesta, ma anche l'ultimo a poter intervenire sulla risposta che viene inviata all'utente.

Ogni middleware offre una funzionalità specifica come ad esempio:

* Intercettare gli errori (ad esempio invocando app.UseDeveloperExceptionPage());
* Servire file statici (invocando app.UseStaticFiles());
* Loggare le richieste;
* Autenticare e autorizzare l'utente;
* Fare routing o riscrivere l'url della richiesta;
* Comprimere le risposte con l'algoritmo Gzip, in modo che richiedano meno banda;
* Regolare l'accesso all'applicazione in base all'indirizzo IP di provenienza;
* ... e così via.

Come regola generale, possiamo dire che **ogni funzionalità applicativa deve essere aggiunta con un middleware**.

**Impostare il nome dell'ambiente nel file launchSettings.json**

Il nome dell'ambiente può essere controllato dal file Properties/launchSettings.json che si trova nel progetto. Impostare opportunamente il nome dell'ambiente ci può permettere di configurare l'applicazione e i suoi middleware in maniera leggermente diversa per assecondare le varie esigenze. Il nome dell'ambiente può essere impostato arbitrariamente ma, di solito, quelli usati sono i seguenti:

* **Development**lo usiamo mentre stiamo sviluppando l'applicazione sul nostro PC o Mac;
* **Staging** lo usiamo durante una fase di test che si svolge subito prima di entrare in produzione. L'applicazione si trova su un server quanto più simile a quello di produzione, per verificare se funziona bene in condizioni diverse da quelle del PC in cui è stata sviluppata;
* **Production** lo usiamo quando l'applicazione è in produzione e viene eseguita sul server, in modo che sia accessibile dai suoi utenti.

**Impostare gli indirizzi di avvio dell'applicazione**

possiamo ridefinire questi indirizzi dal file Properties/launchSettings.json.

"applicationUrl": "https://localhost:7041;http://localhost:5263",

cambiando questi indirizzi qui.

**la directory wwwroot**

è una directory che tiene tutti i file statici che vogliamo usare nella nostra applicazione, perché altrimenti non sarebbero raggiungibili tramite richiesta web

* immagini
* css
* file javascript

i file di log, i database ecc. non si creano in wwwroot, ma devono stare al suo esterno in qualsiasi altra directory, che, ad esempio, possiamo chiamare Data.

Non essendo nella cartella wwwroot siamo certi che non potranno essere scaricati tramite una richiesta web.

esempio: facciamo un file .svg con il nostro logo dell'applicazione e lo chiamiamo logoMC.svg

per poterlo inserire in wwwroot, basta trascinarne dentro una copia

se però ora lanciamo la app non ci troviamo e andiamo all'indirizzo che ci viene espresso sul browser

<http://localhost:7041/>

ed inseriamo

<http://localhost:7041/logoMC.svg>

non appare il logo perché il middleware non è configurato

Per fare questo entriamo nel file Program.cs e inseriamo il metodo .UseStaticFiles() alla variabile app.

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.UseStaticFiles();

app.Run();

ricaricando la richiesta

<http://localhost:7041/logoMC.svg>

adesso compare il logo

REALIZZARE UNA APPLICAZIONE WEB

vedi file "testo della specifica.doc" per introduzione.

1. Poter controllare lo sviluppo dei file della applicazione con **git**