Cheat Sheets

Entity Framework Core

Add-Migration

References

Code First - http://www.entityframeworktutorial.net/efcore/entity-framework-core-console-application.aspx

Migrations Commands

Add-Migration

Creates a migration script analysing changes since the previous model. No modification is done to database.
After generation you can edit the UP and Down from the Migration object.

Install the Package Manager Console tools by running the following command in Package Manager Console:

PowerShell

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

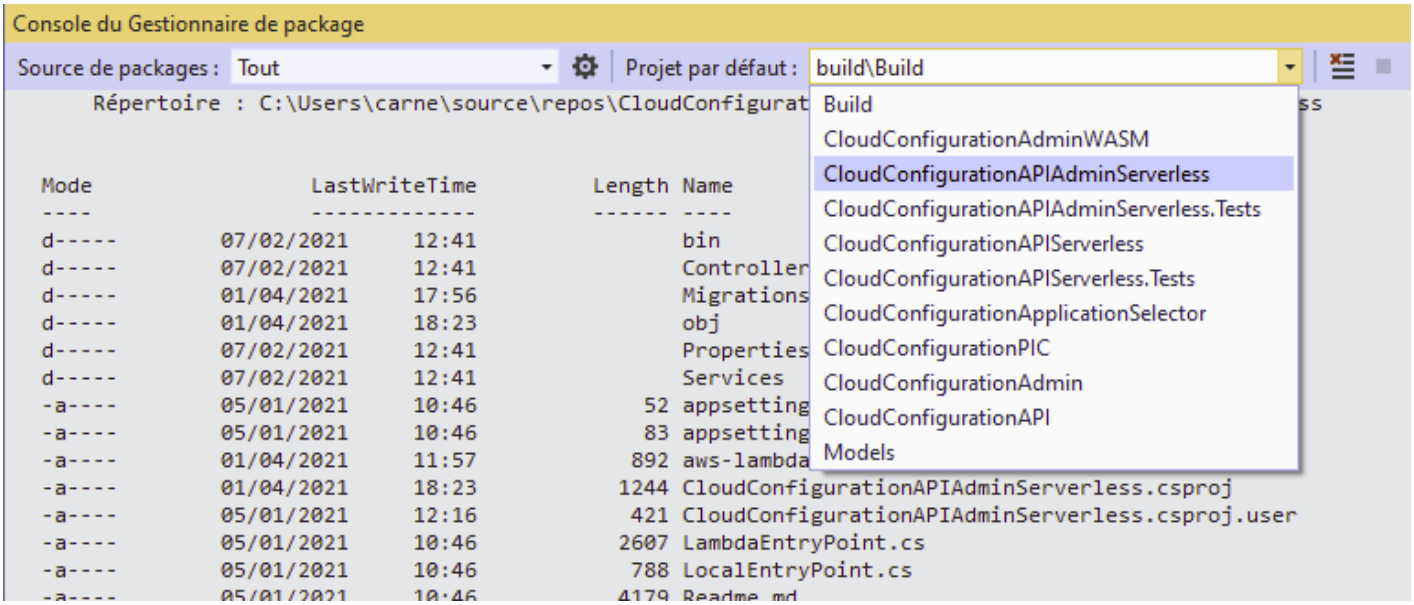
Update the tools by running the following command in Package Manager Console.
PowerShell

```
Update-Package Microsoft.EntityFrameworkCore.Tools
```

Create Database Schema

In Visual Studio, open NuGet Package Manager Console from Tools -> NuGet Package Manager -> Package Manager Console

Check if you are on the correct project :



and enter the following command:

```
PM> add-migration CreateSchoolDB
```

After creating a migration, we still need to create the database using the update-database command in the Package Manager Console, as below.

```
PM> update-database –verbose
```

DBContext

DbContext Methods

Method
Add
AddAsync
AddRange
AddRangeAsync
Attach
AttachRange

Entry
Find
FindAsync
Remove
RemoveRange
SaveChanges
SaveChangesAsync
Set
Update
UpdateRange
OnConfiguring
OnModelCreating

DbContext Properties

Method
ChangeTracker
Database
Model

Scaffold Database (PostGres)

Créer un projet C# ASP Net Core

Il faut ajouter le Nuget correspondant à Postgres dans le projet avec les commandes suivantes

```
Install-Package Microsoft.EntityFrameworkCore.Design
Install-Package Npgsql.EntityFrameworkCore.PostgreSQL
```

Installing the tools

`dotnet ef` can be installed as either a global or local tool. Most developers prefer installing `dotnet ef` as a global tool using the following command:

.NET Core CLI

```
dotnet tool install --global dotnet-ef
```

To use it as a local tool, restore the dependencies of a project that declares it as a tooling dependency using a tool manifest file.

Update the tool using the following command:

.NET Core CLI

```
dotnet tool update --global dotnet-ef
```

Verify installation

Run the following commands to verify that EF Core CLI tools are correctly installed:

.NET Core CLI

```
dotnet ef
```

The output from the command identifies the version of the tools in use:

Output

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\

Entity Framework Core .NET Command-line Tools 2.1.3-rtm-32065

<Usage documentation follows, not shown.>
```

```
dotnet ef dbcontext scaffold
"Server=localhost;Database=Test;Username=postgres;Password=password"
Npgsql.EntityFrameworkCore.PostgreSQL -o Models
```

Ajouter Postgres dans Projet ASP Net Core

Installer les Packages Nuget

Il faut ajouter le Nuget correspondant à Postgres dans le projet avec les commandes suivantes

```
Install-Package Microsoft.EntityFrameworkCore.Design
Install-Package Npgsql.EntityFrameworkCore.PostgreSQL
```

Vérifier que l'on a un DBContext

Il faut au moins un model qui dérive de DBContext nécessaire.

```
public partial class PEActionContext : DbContext
{
  public PEActionContext()
  {
  }

  public PEActionContext(DbContextOptions<PEActionContext> options)
  : base(options)
  {
  }

  public virtual DbSet<Agence> Agences { get; set; }

..... etc
```

En ajoutant un DbSet on rends la table disponible.

Fichier Startup.cs

Dans la méthode **ConfigureServices** ajouter :

```
services.AddDbContext<PEActionContext>(options =>

options.UseNpgsql(

@$"Server={Configuration["DBHost"]};Port=5432;Database={Configuration["DBDatabase"]};Username={
                                        );
```

Cela rajoute au moteur d'injection le DbContext pour qu'il puisse ensuite être utilisé dans le code.

Pour l'utiliser depuis un Controller

Il faut pour cela initialiser le constructeur du Controller avec l'objet DBContext correspondant, dans notre exemple PEActionContext que nous avons déclaré dans le service de Startup.cs

Si le contructeur n'exite pas on le crée avec pour paramètre un objet de type PEActionContext .

C'est le moteur d'injection de ASP Net Core qui se chargera de créer l'instance de l'objet dans le paramètre du constructeur.

```
[Route("api/[controller]")]
    public class AgencesController : ControllerBase
    {
        private PEActionDbContext _peActionDbContext;
        public AgencesController(PEActionDbContext peActionDbContext)
        {
            _peActionDbContext = peActionDbContext;
        }
```

ASP Net Core

Configurer Swagger .Net Core 3.1

Installer les Packages Nuget

Il faut ajouter le Nuget correspondant à Swagger.

```
Install-Package Swashbuckle.AspNetCore
```

Fichier Startup.cs

Dans la section des **using** :

```
using Microsoft.OpenApi.Models;
```

Dans la méthode **ConfigureServices** ajouter :

```
 services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "Cloud Configuration", Version =
"v1" });
            });
```

Dans la méthode **ConfigureServices** ajouter :

```
   if (env.IsDevelopment())
   {
     app.UseSwagger();
     app.UseSwaggerUI(c =>
                      {
```

```
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Cloud Configuration API
V1");
                    });
   }
```

> Attention il y a des problèmes de compatibilité avec OData