**Tác dụng và lợi ích của MigrationsAssembly**

**1 Tách biệt các dự án**: Khi DbContext nằm trong một dự án library riêng (ví dụ như FS.IdentityFramework), nhưng bạn muốn quản lý các migrations trong một dự án khác (ví dụ App.API), MigrationsAssembly cho phép bạn làm điều đó.

**2 Quản lý dễ dàng hơn**: Giúp giữ các migrations cùng với dự án chính (App.API), từ đó bạn có thể dễ dàng tạo, áp dụng, và quản lý migrations mà không cần phải tham chiếu đến dự án chứa DbContext.

**3 Thiết lập đúng cấu trúc thư mục**: Khi bạn chạy lệnh dotnet ef migrations add, các tệp migration sẽ được tạo trong thư mục của dự án được chỉ định, thay vì tạo trong thư mục dự án chứa DbContext.

CẤU HÌNH MÔI TRƯỜNG EF:

Khi bạn sử dụng nhiều `DbContext` trong các dự án khác nhau và muốn chỉ định rõ ràng môi trường khi chạy lệnh `dotnet ef migrations`, bạn cần thực hiện các bước sau để đảm bảo mỗi `DbContext` có thể áp dụng cấu hình môi trường riêng:

### 1. Đặt `ASPNETCORE\_ENVIRONMENT` cho từng `DbContext`

Khi chạy lệnh `dotnet ef migrations` cho từng `DbContext`, bạn sẽ cần chỉ định môi trường thông qua `ASPNETCORE\_ENVIRONMENT` cho từng lệnh. Ví dụ:

```bash

ASPNETCORE\_ENVIRONMENT=Development dotnet ef migrations add MigrationForDbContext1 --project ProjectForDbContext1

ASPNETCORE\_ENVIRONMENT=Production dotnet ef migrations add MigrationForDbContext2 --project ProjectForDbContext2

```

Trong đó:

- `ProjectForDbContext1` và `ProjectForDbContext2` là đường dẫn đến các dự án chứa các `DbContext` khác nhau.

- `MigrationForDbContext1` và `MigrationForDbContext2` là tên migration của mỗi `DbContext`.

### 2. Chỉ định `DbContext` khi thêm Migration

Nếu bạn có nhiều `DbContext` trong cùng một dự án hoặc trong cùng một solution, hãy dùng tham số `--context` để chỉ định `DbContext` cần tạo migration:

```bash

ASPNETCORE\_ENVIRONMENT=Development dotnet ef migrations add MigrationForDbContext1 --context DbContext1 --project ProjectForDbContext1

ASPNETCORE\_ENVIRONMENT=Production dotnet ef migrations add MigrationForDbContext2 --context DbContext2 --project ProjectForDbContext2

```

### 3. Cấu hình môi trường trong từng dự án

Trong mỗi dự án chứa `DbContext`, bạn có thể đảm bảo các tệp `appsettings.json`, `appsettings.Development.json`, và `appsettings.Production.json` được cấu hình đúng theo môi trường tương ứng. EF Core sẽ đọc các tệp cấu hình này dựa trên biến môi trường `ASPNETCORE\_ENVIRONMENT` khi bạn chạy lệnh migration.

### 4. Sử dụng cấu hình riêng cho từng `DbContext`

Nếu mỗi `DbContext` cần các cấu hình riêng biệt, bạn có thể tạo các tệp cấu hình riêng (ví dụ: `appsettings.DbContext1.json`, `appsettings.DbContext2.json`) và tải chúng trong phương thức `OnConfiguring` hoặc `ConfigureServices` để đảm bảo rằng `DbContext` chỉ đọc các cấu hình cần thiết.

Ví dụ:

```csharp

public class DbContext1 : DbContext

{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var environment = Environment.GetEnvironmentVariable("ASPNETCORE\_ENVIRONMENT");

var configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json")

.AddJsonFile($"appsettings.{environment}.json", optional: true)

.AddJsonFile("appsettings.DbContext1.json", optional: true) // Cấu hình riêng

.Build();

optionsBuilder.UseSqlServer(configuration.GetConnectionString("DbContext1Connection"));

}

}

```

### Tóm lại:

1. Sử dụng `ASPNETCORE\_ENVIRONMENT` để chỉ định môi trường khi chạy lệnh migration.

2. Chỉ định rõ ràng `DbContext` và dự án chứa nó với `--context` và `--project`.

3. Sử dụng cấu hình riêng cho từng `DbContext` nếu cần thiết.

Điều này sẽ giúp bạn quản lý cấu hình môi trường cho từng `DbContext` một cách rõ ràng và linh hoạt.