**What is Entity Framework?**

* Entity Framework (EF) is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects.
* It eliminates the need for most of the data-access code that developers usually need to write.
* Its purpose is to abstract the ties to a relational database.

**Why Entity Framework?**

Entity Framework is an ORM, and ORMs are aimed to increase the developer’s productivity by reducing the redundant task of persisting the data used in our applications.

**Features of Entity Framework**

* Entity Framework is a lightweight and extensible object-relational mapping (ORM) technology.
* Entity Framework supports multiple platforms like Windows, Linux, and macOS.
* Entity Framework supports both relational and non-relational data sources.
* Entity Framework works efficiently with widely used databases like SQL Server, SQL Server Compact, SQLite, and PostgreSQL.
* Entity Framework makes it easier for programmers to perform create, read, update, and delete (CRUD) operations by supporting databases. It also makes it easier for developers to perform unit testing by keeping in-memory tables.
* **Entity Framework Development Approaches**
* There are three approaches to creating entity frameworks.

**Code First Approach**

* The Code First approach enables us to create a model and relation using classes and then create the database from these classes. It enables us to work with the Entity Framework in an object-oriented manner. In this approach, you can use empty databases and add tables too.

**Model First Approach**

* In this approach, model classes and their relation are created first using the ORM designer, and the physical database will be generated using this model. The Model First approach means we create a diagram of the entity and relation that will be converted automatically into a code model.

**Database First Approach**

* The Database First approach enables us to create an entity model from the existing database. This approach helps us to reduce the amount of code that we need to write. The following procedure will create an entity model using the Database First approach.

**What is Repository Pattern**

* Repository Pattern is an abstraction of the Data Access Layer. It hides the details of how exactly the data is saved or retrieved from the underlying data source. The details of how the data is stored and retrieved is in the respective repository.
* For example, you may have a repository that stores and retrieves data from an in-memory collection. You may have another repository that stores and retrieves data from a database like SQL Server.

A diagram of a computer

Description automatically generated

**Steps to Follow to Implement EF8 in ASP.NET Web API Project**

**Step 1.** Install Entity Framework Package from NuGet.

Right Click on your project

-> Click on”**Manage NuGet Packages**” -> Open “**Browse**” tab -> search below EF8 package

1. Microsoft.EntityFrameworkCore
2. Microsoft.EntityFrameworkCore.Design
3. Microsoft.EntityFrameworkCore.Tools
4. Microsoft.EntityFrameworkCore.SqlServer

Step2: Add Entity Folder & ApplicationDbContext

using Microsoft.EntityFrameworkCore;  
  
namespace DotNet8WebAPI.Entity  
{  
 public class ApplicationDbContext: DbContext  
 {  
 public ApplicationDbContext (DbContextOptions< ApplicationDbContext > options) : base(options)  
 {  
 }  
 }  
}

Step 3: Registered DB Model in ApplicationDbContext.cs

Step 4: Perform Data Seeding if required

Step 5: Add ConnectionStrings in appsettings.json file

"ConnectionStrings": {

"DefaultConnection:”Pass Your Connection String”

}

Step 6: Register ApplicationDbContext

Select your Database, I’m using an SQL server as a database. That’s why I called the **UseSqlServer** method. **Provide ConnectionString in appSettings.json**

builder.Services.AddDbContext<ApplicationDbContext>(db => db.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")), ServiceLifetime.Singleton);

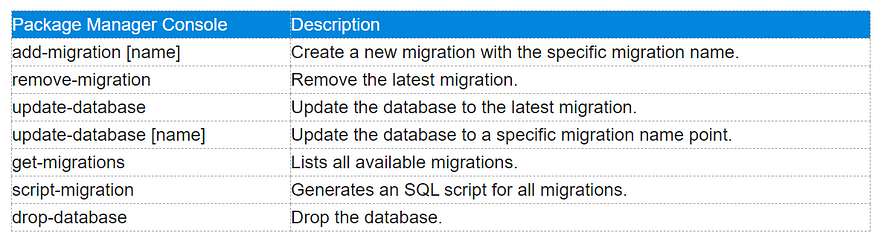
Step 7: Inject ApplicationDbContext in Services

Perform all CRUD operations with async and await for all DB operations.

Step 8: Update IService class with Task<>

Step 9: Implement async-await in all action methods in controllers

Add-migrations and update-database



Step 10: Test APIs

**Authentication**

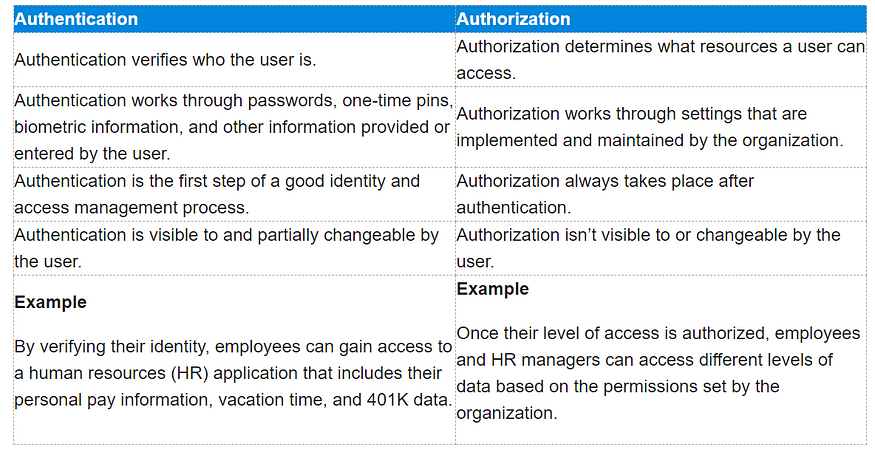
Authentication is knowing the identity of the user.

**For example**: Alice logs in with her username and password, and the server uses the password to authenticate Alice.

**Authorization**

Authorization is deciding whether a user is allowed to act.

**For example**: Alice has permission to get a resource but not create a resource.



Step 1: Add JwtBearer package from Microsoft:

Install **Microsoft.AspNetCore.Authentication.JwtBearer** library to implement the JWT token in our application.

Step 2: Add JWT Middleware

Add Jwt Middleware in our app. For this, please follow the below steps.

* Create a Helpers folder in an API solution
* Add a class called **“JwtMiddleware”**
* Add the JwtMiddleware constructor and inject RequestDelegate and AppSettings in the implement **Invoke** method
* inside invoke method read the **Authorization** token from the current request and forward to the **attachUserToContext** method for **validation** and extract user info and attach to the current request.
* Add our “JwtMiddleware” to our app
* for this, go to Program.cs and add it.