Udemy course: Ultimate ASP.NET pt. 3

Database Modelling and Entity

#14: Setup Entity Framework in API Project

- packagename
  - Microsoft.EntityFrameworkCore.SqlServer
  - dotnet add package <name>
  - also install Microsoft.EntityFrameworkCore.Tools
- course narrator puts it into a MS Sql Database
- Note: about appsettings.json new entry:

  -
    ```
    "ConnectionStrings": {
        "HotelListingDbConnectionString":
        "Server=(localdb)\\mssqllocaldb;Database=HotelListingAPIDb;Trusted_Connection=True;MultipleActiveResultSets=True"
        },
    ```
  - localdb is probably specific to Visual Studio! builtin server of Visual Studio!
  - MultipleActiveResultSets is saying you might have multiple simulationeous connections from the app to the database
- a way to connect the database
  - in Program.cs
    - create a connectionString

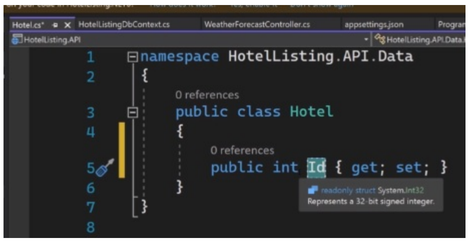
    -
      ```
      var
      connectionString
      =
      builder.Configuration.GetConnectionString("HotelListingDbConnectionString");
          builder.Services.AddDbContext<HotelListingDbContext>(options
          => {
          options.UseSqlServer(connectionString);
          });
      ```
    - Note: at this point `HotelListingDbContext` does not exist yet
    - ConnectionString is used in options
    - if you use another database, use anther option in place of UseSqlServer!
- define a new class in Data folder (part of the model)
  - class name example HotelListingDbContext
  - inherit from DbContext
  - with import:
    ```
    using Microsoft.EntityFrameworkCore;
    ```
  - Tipp:
    - create a constructor fast inside the class with
    - ctor tab
  - this class is like the contract between our app and the database
  - the constructor has the options from the builder
  - constructor code:

    -
      ```
      public HotelListingDbContext(DbContextOptions options) : base(options)
      {
      }
      ```
  - this is were we will be outlining the different tables and setting different configurations
  - rules, contract for our database!
- we add the namespaces of EntityFrameworkCore and of our class in Program.cs

  -
    ```
    using HotelListingAPI.VSCode.Data;
    using Microsoft.EntityFrameworkCore;
    ```

#15 Implement Data Classes and Perform Migrations

- SQL Management Studio
  - alternative 1: Azure Data Studio
    https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver16
  - alternative 2:
    - connect from Visual Studio Code to SQL Server on Azure
      tutorial: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-connect-query-vscode
  - or you can use Visual Studio Code to connect to a SQL Server instance
    - a SQL Server instance in preview version, can run on Linux also
    - read more on:
      https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-overview?view=sql-server-ver16
- we are doing a Hotel Listing API
  - list of hotels
  - hotels and countries
- we keep it simple, so that we can understand the fundamentals of API development
- from these fundamentals you can have more complex applications
- we are doing code first
  - as opposed to database first
  - the course narrator recommends:
    - model the database on a piece of paper first by hand!
    - vizualize the data, visualize the data points for each entity and so on
    - and then you either go write the code or go to create the database
    - after years of experience you might be able to skip the paper draft, for less compex database designs
    - with code first, it gives you more flexibility to change your
- first lean draft of a model:

- 

```
namespace HotelListing.API.Data
{
    0 references
    public class Hotel
    {
        0 references
        public int Id { get; set; }
    }
}
```

readonly struct System.Int32
Represents a 32-bit signed integer.

  - from that the entity framework knows this is a autoincrementing primary key
  - we will have a foreign key in hotel of countries
  - our full Hotel.cas code, after writing this data model down:
    - Note: the Country entity we have to define next

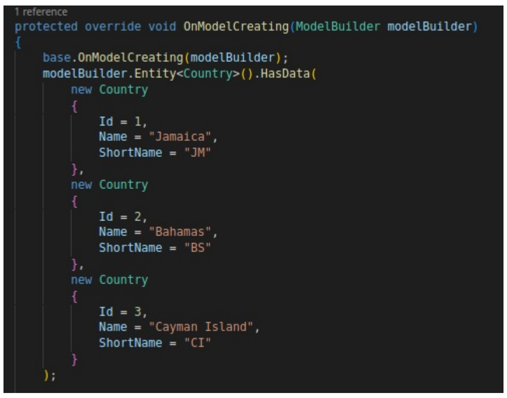
    -
      ```
      public class Hotel

      {

      public int Id { get; set; }

      public string Name { get; set; }

      public string Address { get; set; }

      public double Rating { get; set; }


      [ForeignKey(nameof(CountryId))]

      public int CountryId { get; set; }

      public Country Country { get; set; }

      }
      ```
    - Note: to keep it strongly typed, we use nameof keyword instead of putting the string "CountryId"
    - so it checks the types for us
    - prop tab can be used to create properties!
    - all of this three lines are to be put, to define Country as foreign key
- the other entity /table we will call country
  - the new special field, one country can have many hotels, so...
  - we can put that relation as:

    -
      ```
      public virtual IList<Hotel> Hotels { get; set; }
      ```
  - alternative: ICollection, or HashSets
- about the nullable flag in csproj file
  - when we set it to disable: it will not warn for null variables (variables which are not initialized)
  - but we can keep it on
- in the file of HotelListingDbContext.cs we have to let the class know about the database tables /entities
- to create the database from the code
  - in Visual Studio we would go to the package manager console
    - add-migration InitialMigration
  - in Visual Studio Code you can use your terminal, from there:
    dotnet ef migrations add InitialMigration
    - where InitialMigration is just the migrationname
    - see: https://stackoverflow.com/questions/41536603/visual-studio-code-entity-framework-core-add-migration-not-recognized
  - Note: in current version you have to install dotnet-ef first!
    dotnet tool install --global dotnet-ef
  - this command create migration files for creating the database
    - see Migration folder
    - in the generated files you can find also primary keys and short name
  - to know about more details of the Entity framework...
    - get the full course from the course teacher about it...!
  - it is possible to undo a migration
    - all within Down is undoing the migrations
  - and then execute those migrations with:

    -
      ```
      dotnet ef database update
      ```
    - or in
      Visual Studio: update-database
- In Azure Data Studio:
  - as alternative to Visual Studio server:
  - Create a connection
  - as server put "`(localdb)\mssqllocaldb`"
    - Note: here only one slash
    - localdb is not available for linux or Mac!
    - read: https://stackoverflow.com/questions/45860851/localdb-is-not-supported-on-this-platform
    - or try it with the Azure tutorial above

#16: Bring Seed Data into Tables

- in Data/HotelListingDbContent.cs
  - Method "OnModelCreating"
  - to insert some default data
- example:

- 

```
1 reference
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Country>().HasData(
        new Country
        {
            Id = 1,
            Name = "Jamaica",
            ShortName = "JM"
        },
        new Country
        {
            Id = 2,
            Name = "Bahamas",
            ShortName = "BS"
        },
        new Country
        {
            Id = 3,
            Name = "Cayman Island",
            ShortName = "CI"
        }
    );
```

- see Note about Azure Getting Started
  - connect to SQL Database
  - connection string in appsettings.json could look similar to:

```
"HotelListingDbConnectionString":
"Server=tcp:myazureserver1.database.windows.net;Database=AzureDB;Persist
Security Info=False;User
ID=wolfi;Password=<mypasswordconfidential!>;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;"
```

  - but Note: password better use environment variables instead of putting it there and on Github!!!
    - find better way of putting password!
  -