BUILDING THE BLOG – STEP 1

Code-First Database Development with Entity Framework 6

This document walks you through the steps of setting up a Code-First database design using Entity Framework 6. The examples here link database tables directly to those generated by Entity Framework

To create the blog you will need to add two additional models. Blogpost and Comments.

To get started follow these instructions

1. Create a new MVC Web application in Visual Studio
2. Change the Web.config file in your MVC project so the connection string points to the database you wish to reference (such as your local SQL Express installation or online resource).

   Below is the default connection string that is created by Visual Studio.

```
<connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)
\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-
Blog-20170221050905.mdf;Initial Catalog=aspnet-Blog-20170221050905;Integrated
Security=True" providerName="System.Data.SqlClient" />
  </connectionStrings>
```

   Ask your instructor for the connection string for Azure and make the changes as needed

3. Enable automatic migrations for your project. You will need to open the Package Manager Console. In Visual Studio click on View..Other Windows..Package Manager Console.

   In the Package Manager Console window:
   Enable-Migrations -EnableAutomaticMigrations

   If it works you should something like the following:

```
PM> Enable-Migrations -EnableAutomaticMigrations
Checking if the context targets an existing database...
```

```
Code First Migrations enabled for project Blog.
```

4. Code the classes that will represent your data model.
   You need to create a Comment.cs and BlogPost.cs file in the ***Models*** Folder. Alternatively, you can create a BlogModels.cs and put the BlogPost and Comment Class in the same file.

   The examples below demonstrate the classes needed for the Blog Application.

   NOTE: DO NOT CUT AND PASTE THE CODE IN YOUR APPLICATON. Type the code using the editor.
   You will learn more by doing and improve your ability to solve common compiler issues.

   File: *Comment.cs*

```
using System;
using System.Collections.Generic;

namespace Blog.Models
{

        public class Comment
        {
                public int Id { get; set; }
                public int PostId { get; set; }
                public string AuthorId { get; set; }
                public string Body { get; set; }
                        public DateTimeOffset Created { get; set; }
                        public DateTimeOffset? Updated { get; set; }
                        public string UpdateReason { get; set; }

                        public virtual ApplicationUser Author { get; set; }
```

   File: *BlogPost.cs*

```
using System;
using System.Collections.Generic;

namespace Blog.Models
{
        public class BlogPost
        {
                public BlogPost()
                        {
                        this.Comments = new HashSet<Comment>();
                }

                public int Id { get; set; }
                public DateTimeOffset Created { get; set; }
                public DateTimeOffset? Updated { get; set; }
                public string Title { get; set; }
                public string Slug { get; set; }
                public string Body { get; set; }
                public string MediaURL { get; set; }
                public bool Published { get; set; }

                public virtual ICollection<Comment> Comments { get; set; }
        }
```

5. After the data classes are created, we will need to add attributes to the ApplicationUser class: FirstName, LastName, DisplayName. We will use these to link our blog classes to user entities managed through Entity Framework.

6. Now Add the DbSet declarations for each table to the ApplicationDbContext class.

File: *IdentityModels.cs*

```
public class ApplicationUser : IdentityUser
{
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string DisplayName { get; set; }

        public ApplicationUser()
        {
                this.BlogComments = new HashSet<Comment>();
        }

        public virtual ICollection<Comment> BlogComments { get; set; }

        . . .
}

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
        public ApplicationDbContext()
             : base("DefaultConnection", throwIfV1Schema: false) {  }

        public static ApplicationDbContext Create()
        {
                return new ApplicationDbContext();
        }

        public DbSet<BlogPost> Posts { get; set; }
        public DbSet<Comment> Comments { get; set; }
}
```

7. Now modify the Seed method (found in *Migrations\Configuration.cs*) to seed your database with an initial user and the required roles.
   Note: You may need to add several using statements to the top of the file

   File: *Configuration.cs*

```
protected override void Seed(ApplicationDbContext context)
{
        var roleManager = new RoleManager<IdentityRole>(
                new RoleStore<IdentityRole>(context));

        if (!context.Roles.Any(r => r.Name == "Admin"))
        {
                roleManager.Create(new IdentityRole { Name = "Admin" });
        }
```

8. Run a code-first migration to create/update your database.
   a. In the Package Manager Console window:

      Update-Database

9. Test your application by logging in with the user you have just seeded.
10. Now you are ready to beginning scaffolding controllers and views for your models and securing your application.