CS6004NT
Application Development

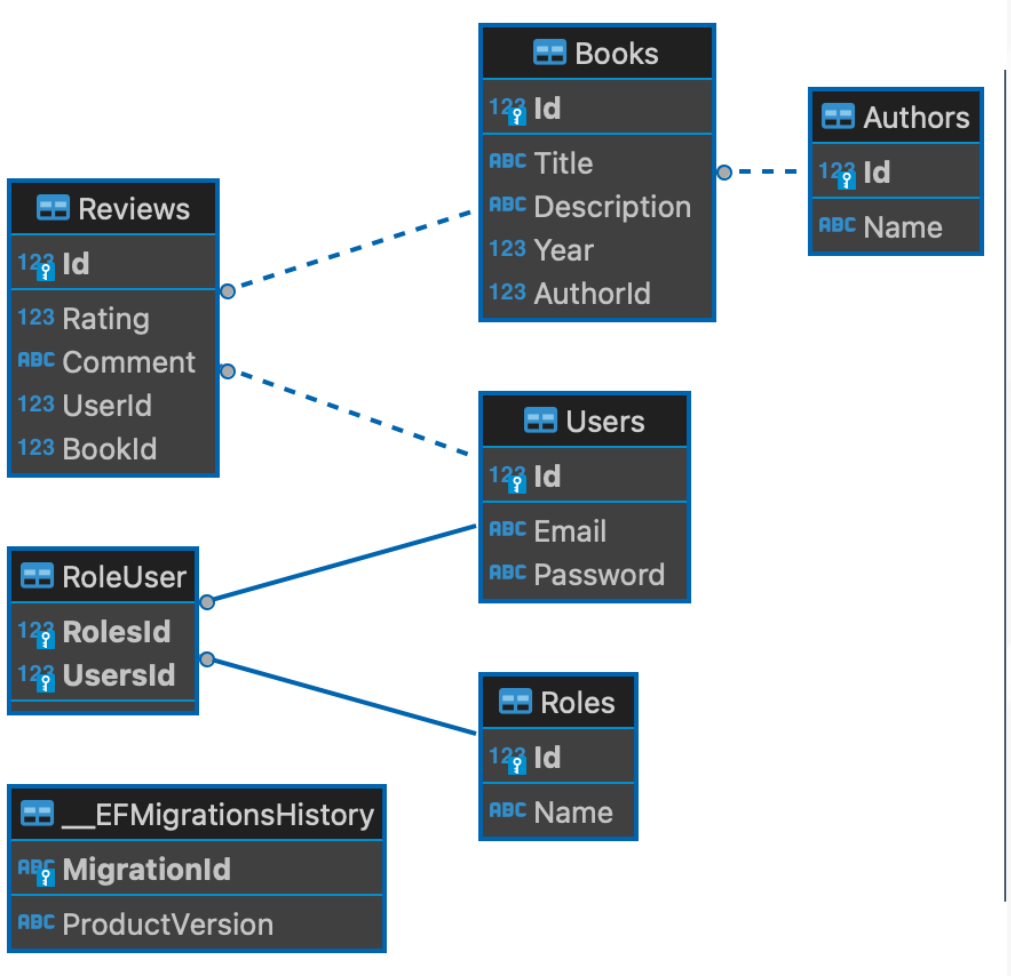
WEEK - 05

EF Core

Entity Framework Core

EF Core

Table / Entity Relationship




```cs
namespace BookReview.Api.Domain.Models;
public class Author
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public ICollection<Book>? Books { get; set; }
}
```

```cs
namespace BookReview.Api.Domain.Models;
public class Book
{
    public int Id { get; set; }
    public string Title { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
    public int Year { get; set; }
    public int AuthorId { get; set; }
    public Author? Author { get; set; }
    public IEnumerable<Review>? Reviews { get; set; }
}
```

EF Core

Table / Entity Relationship and Constraints using Fluent API

BookReviewDbContext.cs:

```
1.  public class BookReviewDbContext : DbContext
2.  {
3.      ... // DbContext Constructors and DbSets
4.      protected override void OnModelCreating(
5.          ModelBuilder modelBuilder
6.      )
7.      {
8.          modelBuilder.Entity<Book>()
9.              .HasMany(b => b.Reviews)
10.             .WithOne(r => r.Book);

11.         modelBuilder.Entity<Book>()
12.             .HasOne(b => b.Author)
13.             .WithMany(a => a.Books)
14.             .HasForeignKey(b => b.AuthorId);
15. ...
```

```
17.         modelBuilder.Entity<Book>()
18.             .HasMany(b => b.Reviews)
19.             .WithOne(r => r.Book);

20.         modelBuilder.Entity<Book>()
21.             .HasOne(b => b.Author)
22.             .WithMany(a => a.Books)
23.             .HasForeignKey(b => b.AuthorId);

24.         modelBuilder.Entity<Book>()
25.             .Property(b => b.Title)
26.             .IsRequired()
27.             .HasMaxLength(100);

28.         modelBuilder.Entity<Book>()
29.             .HasIndex(b => b.Title)
30.             .IsUnique();
31. ...
```

```
        modelBuilder.Entity<Book>()
            .Property(b => b.Description)
            .HasMaxLength(3000);

        modelBuilder.Entity<Author>()
            .HasIndex(a => a.Name)
            .IsUnique();

        modelBuilder.Entity<Review>()
            .Property(r => r.Comment)
            .HasMaxLength(1000);

        modelBuilder.Entity<Review>()
            .Property(r => r.Rating)
            .IsRequired()
            .HasPrecision(2, 1) // up to 2 digits in total, 1 of which should be for the decimal places
            .HasConversion(
                v => Math.Round(v * 2, MidpointRounding.AwayFromZero) / 2, // rounds to the nearest 0.5 value
                v => v) // as it is from provider
            .HasConversion(
                v => v < 0 ? 0 : v > 5 ? 5 : v, // to make sure values are between 0 and 5, inclusive
                v => v)
            .HasDefaultValue(0);
        ...

        modelBuilder.Entity<Role>()
            .HasIndex(r => r.Name)
            .IsUnique();

        modelBuilder.Entity<User>()
            .HasIndex(u => u.Email)
          ...
    }
}
```

EF Core

Managing the Migrations

1. **Create migration**

```
dotnet ef migrations add InitialCreate --output-dir Infrastructure/Migrations/
```

1. **Apply any pending migrations to the DB**

```
dotnet ef database update
```

1. **Create new migration after making changes**

```
dotnet ef migrations add UpdateEntityClasses
```

1. **Remove last migration**

```
dotnet ef migrations remove

dotnet ef migrations remove -f # remove applied migration
```

1. **List of all the available migrations with the status**

```
dotnet ef migrations list
```

EF Core

Migration Script

```
1.  namespace BookReview.Api.Infrastructure.Data.Repositories.Migrations
2.  {
3.      public partial class InitialCreate : Migration
4.      {
5.          protected override void Up(MigrationBuilder migrationBuilder)
6.          {
7.              migrationBuilder.CreateTable(
8.                  name: "Books",
9.                  columns: table => new
10.                 {
11.                     Id = table.Column<int>(type: "integer", nullable: false)
12.                         .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
13.                     ...
14.                     Year = table.Column<int>(type: "integer", nullable: false)
15.                 },
16.                 constraints: table =>
17.                 {
18.                     table.PrimaryKey("PK_Books", x => x.Id);
19.                 });
20.         }

21.         protected override void Down(MigrationBuilder migrationBuilder)
22.         {
23.             migrationBuilder.DropTable(name: "Books");
24.         }
25.     }
26. }
```

EF Core

Logging SQL Query

```
1 builder.Services.AddDbContext<AppDbContext>(options =>
2 {
3     options.UseSqlServer(builder.Configuration.GetConnectionString("Default"));
4     options.EnableSensitiveDataLogging();
5 });
```



```
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.3 initialized 'AppDbContext' using provider 'Microsoft.EntityFrame
workCore.SqlServer:6.0.3' with options: SensitiveDataLoggingEnabled
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (5ms) [Parameters=[@__id_0='6'], CommandType='Text', CommandTimeout='30']
      SELECT TOP(1) [a].[AuthorId], [a].[Name], [a].[WebUrl]
      FROM [Authors] AS [a]
      WHERE [a].[AuthorId] = @__id_0
```

EF Core

Logging SQL Query

```
1      public Task<List<Author>> FilterByPartialNameAsync(string searchString)
2      {
3          var query = _context.Authors.Where(x => x.Name.ToLower().Contains(searchString.ToLower()));
4
5          Console.WriteLine(query.ToQueryString());
6
7          return query.ToListAsync();
8      }
```

```
DECLARE @__ToLower_0 nvarchar(4000) = N'dan';

SELECT [a].[AuthorId], [a].[Name], [a].[WebUrl]
FROM [Authors] AS [a]
WHERE (@__ToLower_0 LIKE N'') OR (CHARINDEX(@__ToLower_0, LOWER([a].[Name])) > 0)
```

EF Core

Raw SQL Queries

```
1     public Task<List<Book>> GetBooksByAuthorAsync(string authorId)
2     {
3  1      return _context.Books.FromSqlRaw($"SELECT * FROM dbo.Books WHERE AuthorId = {authorId}").ToListAsync();
4     }
5
6     public Task<List<Book>> GetBooksByAuthorAsync(string authorId)
7     {
8  2      return _context.Books.FromSqlRaw("SELECT * FROM dbo.Books WHERE AuthorId = {0}", authorId).ToListAsync();
9     }
10
11    public Task<List<Book>> GetBooksByAuthorAsync(string authorId)
12    {
13 3      return _context.Books.FromSqlInterpolated($"SELECT * FROM dbo.Books WHERE AuthorId = {authorId}").ToListAsync();
14    }
```

EF Core

Querying data from related entities

```
1.  // Using Include method for Navigation Property
2.  List<Book> result = await _dbContext.Books
3.      .Include(x => x.Author)
4.      .Include(x => x.Reviews)
5.      .ToListAsync();
6.  // Using joins in query syntax
7.  List<Book> result = (from book in _dbContext.Books
8.      join author in _dbContext.Authors on book.AuthorId equals author.Id
9.      join review in _dbContext.Reviews on book.Id equals review.BookId into reviews
10.     select new Book {
11.         Id = book.Id,
12.           Title = book.Title,
13.           Year = book.Year,
14.           Author = author,
15.           Reviews = reviews.Select(r => new Review {
16.             Rating = r.Rating
17.           })
18.       })
19.       .ToListAsync();
```

EF Core

Querying data from related entities

```
1. // Using joins in method syntax
2. var result = await _dbContext.Books
3.     .Join(_dbContext.Authors, book => book.AuthorId, author => author.Id, (book, author) => new { book, author })
4.     .GroupJoin(_dbContext.Reviews,
5.                x => x.book.Id,
6.                review => review.BookId,
7.                (bookAuthor, reviews) => new Book
8.                {
9.                    Id = bookAuthor.book.Id,
10.                   Title = bookAuthor.book.Title,
11.                   Year = bookAuthor.book.Year,
12.                   Author = bookAuthor.author,
13.                   Reviews = reviews.ToList()
14.               })
15.    .ToListAsync();
```

EF Core

Using AsNoTracking for Read Performance

Making `AsNoTracking` the default behaviour for all DB read query

```
builder.Services.AddDbContext<AppDbContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("Default"));
    options.EnableSensitiveDataLogging();
    options.UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking);
});
```

Example of when Tracking is required

```
    public async Task UpdateNameAsync(int id, string newName)
    {
        var author = await _context.Authors.AsTracking().FirstOrDefaultAsync(r => r.AuthorId == id);

        if (author is not null)
        {
            author.Name = newName;
            await _context.SaveChangesAsync();
        }
    }
```

EF Core

Single vs Split Query

```
1     public Task<List<Book>> GetAllAsync()
2     {
3         return _context.Books
4         .Include(x => x.Authors)
5         .Include(x => x.Reviews)
6             .ThenInclude(x => x.AppUser)
7         .ToListAsync();
8     }
```

```
SELECT [b].[BookId], [b].[Description], [b].[IsDeleted], [b].[PublishedOn], [b].[Title], [t].[AuthorId], [t].[Name],
      [t].[WebUrl], [t].[AuthorsAuthorId], [t].[BooksBookId], [t0].[ReviewId], [t0].[AppUserId], [t0].[BookId],
      [t0].[Comment], [t0].[AppUserId0], [t0].[Email], [t0].[Name]
FROM [Books] AS [b]
LEFT JOIN (
    SELECT [a0].[AuthorId], [a0].[Name], [a0].[WebUrl], [a].[AuthorsAuthorId], [a].[BooksBookId]
    FROM [AuthorBook] AS [a]
    INNER JOIN [Authors] AS [a0] ON [a].[AuthorsAuthorId] = [a0].[AuthorId]
) AS [t] ON [b].[BookId] = [t].[BooksBookId]
LEFT JOIN (
    SELECT [r].[ReviewId], [r].[AppUserId], [r].[BookId], [r].[Comment], [a1].[AppUserId] AS [AppUserId0],
      [a1].[Email], [a1].[Name]
    FROM [Reviews] AS [r]
    INNER JOIN [AppUsers] AS [a1] ON [r].[AppUserId] = [a1].[AppUserId]
) AS [t0] ON [b].[BookId] = [t0].[BookId]
ORDER BY [b].[BookId], [t].[AuthorsAuthorId], [t].[BooksBookId], [t].[AuthorId], [t0].[ReviewId]
```

	BookId	Description	IsDeleted	PublishedOn	Title	AuthorId	Name	WebUrl	AuthorsAuthorId	BooksB
96	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
97	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
98	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
99	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
100	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
101	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
102	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
103	1	While in Paris on b	0	03-23 15:30:36.103	The Da Vir	6	Dan Brown	https://danbrown.com	6	
104	2	Nearly 30 years ag	0	03-23 15:37:33.853	Surreal Nu	10	Donald Knuth	https://www-cs-faculty	10	
105	2	Nearly 30 years ag	0	03-23 15:37:33.853	Surreal Nu	10	Donald Knuth	https://www-cs-faculty	10	

EF Core

Single vs Split Query

```
1      public Task<List<Book>> GetAllAsync()
2      {
3          return  context.Books
4          .AsSplitQuery()
5          .Include(x => x.Authors)
6          .Include(x => x.Reviews)
7              .ThenInclude(x => x.AppUser)
8          .ToListAsync();
9      }
```

```
-- Books
  SELECT [b].[BookId], [b].[Description], [b].[IsDeleted], [b].[PublishedOn], [b].[Title]
  FROM [Books] AS [b]
  ORDER BY [b].[BookId]
-- Authors
  SELECT [t].[AuthorId], [t].[Name], [t].[WebUrl], [b].[BookId]
  FROM [Books] AS [b]
  INNER JOIN (
      SELECT [a0].[AuthorId], [a0].[Name], [a0].[WebUrl], [a].[BooksBookId]
      FROM [AuthorBook] AS [a]
      INNER JOIN [Authors] AS [a0] ON [a].[AuthorsAuthorId] = [a0].[AuthorId]
  ) AS [t] ON [b].[BookId] = [t].[BooksBookId]
  ORDER BY [b].[BookId]
-- Reviews
  SELECT [t].[ReviewId], [t].[AppUserId], [t].[BookId], [t].[Comment], [t].[AppUserId0], [t].[Email], [t].[Name], [b].[BookId]
  FROM [Books] AS [b]
  INNER JOIN (
      SELECT [r].[ReviewId], [r].[AppUserId], [r].[BookId], [r].[Comment], [a].[AppUserId] AS [AppUserId0], [a].[Email], [a].[Name]
      FROM [Reviews] AS [r]
      INNER JOIN [AppUsers] AS [a] ON [r].[AppUserId] = [a].[AppUserId]
  ) AS [t] ON [b].[BookId] = [t].[BookId]
  ORDER BY [b].[BookId]
```

	BookId	Description	IsDeleted	PublishedOn	Title
1	1	While in Paris on business, Harvard sy	0	2022-03-23 15:30:36.103	The Da Vinci Code
2	2	Nearly 30 years ago, John Horton Cor	0	2022-03-23 15:37:33.853	Surreal Numbers

	AuthorId	Name	WebUrl	BookId
1	6	Dan Brown	https://danbrown.com	1
2	10	Donald Knuth	https://www-cs-faculty	2

	ReviewId	AppUserId	BookId	Comment	AppUserId0	Email	Name	BookId
1	1	1	1	Good review for book 1	1	john@doe.com	John Doe	1
2	3	2	1	Good review for book 1	2	jane@doe.com	Jane Doe	1
3	5	2	1	Comment 99	2	jane@doe.com	Jane Doe	1
4	6	2	1	Comment 0	2	jane@doe.com	Jane Doe	1
5	7	2	1	Comment 1	2	jane@doe.com	Jane Doe	1
6	8	2	1	Comment 2	2	jane@doe.com	Jane Doe	1
7	9	2	1	Comment 3	2	jane@doe.com	Jane Doe	1
8	10	2	1	Comment 4	2	jane@doe.com	Jane Doe	1
9	11	2	1	Comment 5	2	jane@doe.com	Jane Doe	1
10	12	2	1	Comment 6	2	jane@doe.com	Jane Doe	1

Islington college (इस्लिङ्टन कलेज)
informa college

EF Core

Global Query Filter

Use `IsDeleted` property to mark a row as soft-deleted

```
1     public async Task DeleteAsync(int id)
2     {
3         var book = await _context.Books.AsTracking().FirstOrDefaultAsync(r => r.BookId == id);
4
5         if (book is not null)
6         {
7             book.IsDeleted = true;
8             await _context.SaveChangesAsync();
9         }
10    }
```

AuthorsService method

Exclude soft-deleted rows from read query result

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Book>().HasQueryFilter(x => !x.IsDeleted);
}
```

AppDbContext overrride method

Use `.IgnoreQueryFilters()` operator to ignore the Global Query Filter

ASP.NET Core Identity

Authentication and Authorization

ASP.NET Core Identity

Setting Up ASP.NET Core Identity

1. Add NuGet Pacakge:

```
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore --version 6.0.15
```

1. Update BookReviewDbContext.cs:

```
2.  namespace BookReview.Api.Infrastructure.Data;
3.  public class BookReviewDbContext : IdentityDbContext<IdentityUser, IdentityRole, string>
4.  {
5.      ...
6.      protected override void OnModelCreating(ModelBuilder modelBuilder)
7.      {
8.          base.OnModelCreating(modelBuilder);
9.          ...
10.         modelBuilder.Entity<Review>()
11.             .HasOne<IdentityUser>()
12.             .WithMany()
13.             .HasForeignKey(r => r.UserId)
14.             .IsRequired();
15.     }
16. }
```

ASP.NET Core Identity

Setting Up ASP.NET Core Identity

3. Program.cs:

```
1.  var builder = WebApplication.CreateBuilder(args);
2.  ...
3.  builder.Services.AddIdentity<IdentityUser, IdentityRole>(options => {
4.          options.SignIn.RequireConfirmedAccount = false;
5.          ...
6.      })
7.      .AddEntityFrameworkStores<BookReviewDbContext>();
8.  ...
9.  var app = builder.Build();
10. ...
11. app.UseAuthentication();
12. ...
13. app.Run();
```

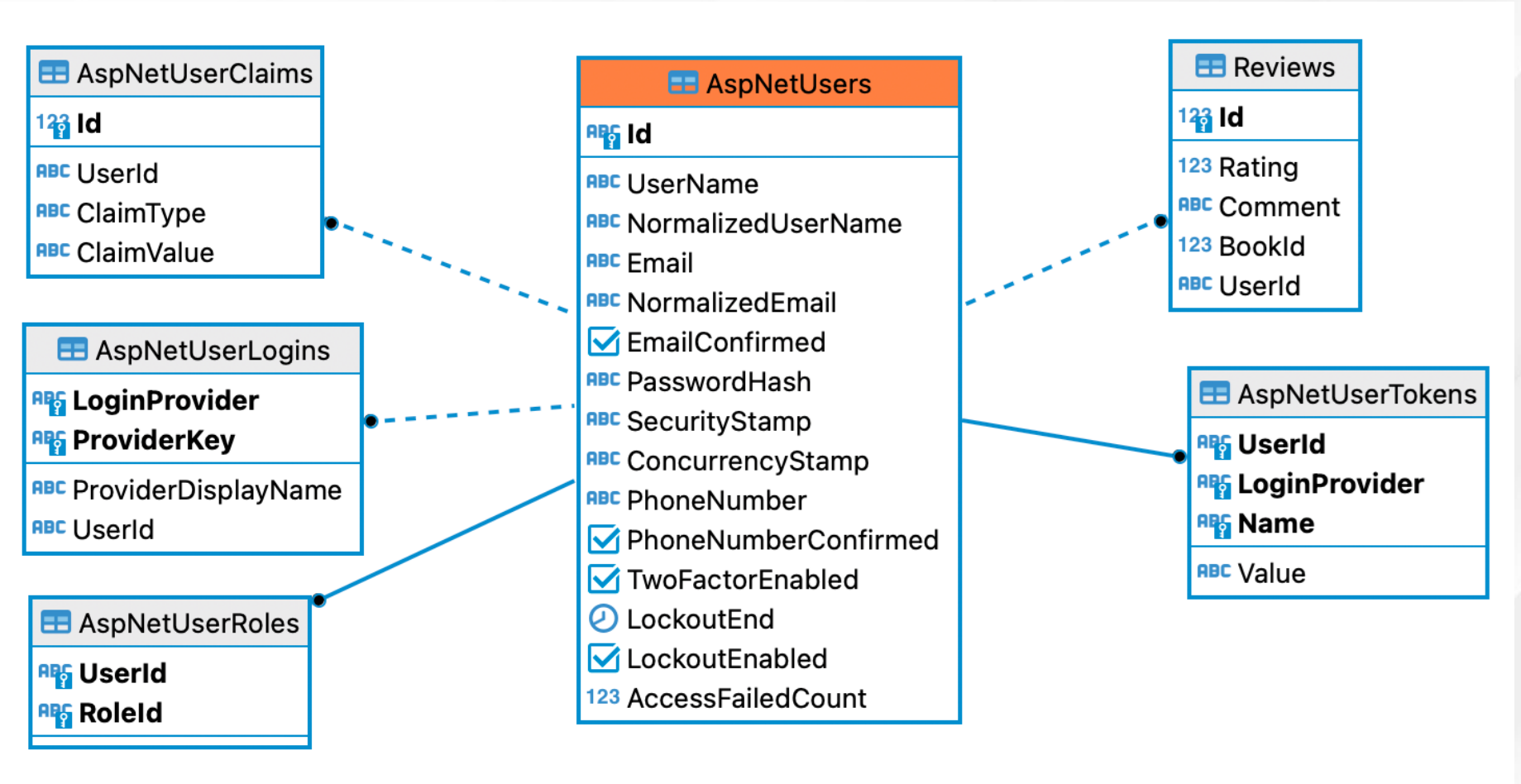
3. Add and Apply Migration:

```
dotnet ef migrations add AddIdentity

dotnet ef database update
```

ASP.NET Core Identity

ASP.NET Core Identity Entity Relation



ASP.NET Core Identity

Seeding Roles and Users

SeedIdentityData.cs:

```
1.  namespace BookReview.Api.Infrastructure.Identity;
2.  public static class SeedIdentityData
3.  {
4.      public static async Task InitializeAsync(IServiceProvider services)
5.      {
6.          var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
7.          var userManager = services.GetRequiredService<UserManager<IdentityUser>>();

8.          // Create roles
9.          if (!await roleManager.RoleExistsAsync("Admin"))
10.         {
11.             await roleManager.CreateAsync(new IdentityRole("Admin"));
12.         }
13.         if (!await roleManager.RoleExistsAsync("User"))
14.         {
15.             await roleManager.CreateAsync(new IdentityRole("User"));
16.         }
17. ...
```

```
        // Create admin user
        var adminUser = new IdentityUser
        {
            UserName = "admin@user.com",
            Email = "admin@user.com",
            EmailConfirmed = true
        };
        var adminPassword = "Admin123#";
        if (await userManager.FindByEmailAsync(adminUser.Email) == null)
        {
            var result = await userManager.CreateAsync(adminUser, adminPassword);
            if (result.Succeeded)
            {
                await userManager.AddToRoleAsync(adminUser, "Admin");
            }
        }
    }
}
```

ASP.NET Core Identity

Seeding Roles and Users

Program.cs:

```
1. var app = builder.Build();


2. // Seed data
3. using var scope = app.Services.CreateScope();
4. var services = scope.ServiceProvider;
5. await SeedIdentityData.InitializeAsync(services);
```

ASP.NET Core Identity

User Login

AuthController.cs:

```
1.  namespace BookReview.Api.Controllers;

2.  [ApiController]
3.  public class AuthController : ControllerBase
4.  {
5.      private readonly SignInManager<IdentityUser> _signInManager;
6.      private readonly UserManager<IdentityUser> _userManager;
7.      public AuthController(SignInManager<IdentityUser> signInManager, UserManager<IdentityUser> userManager)
8.      ...
9.      [HttpPost("/api/auth/login")]
10.     public async Task<IActionResult> LoginAsync([FromBody] LoginRequest login)
11.     {
12.         var result = await _signInManager.PasswordSignInAsync(login.Username, login.Password, true, lockoutOnFailure: false);
13.         if (result.Succeeded)
14.             return Ok();
15.         return Unauthorized();
16.     }
17. ...
```

ASP.NET Core Identity

User Profile

Program.cs:

```
1.  [Authorize]
2.  [HttpGet("/api/auth/profile")]
3.  public async Task<ActionResult<UserResponse>> GetProfileAsync()
4.  {
5.      var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
6.      var user = await _userManager.FindByIdAsync(userId);
7.      var userResponse = new UserResponse
8.      {
9.          Id = user.Id,
10.         Email = user.Email,
11.         EmailConfirmed = user.EmailConfirmed
12.     };

13.     return Ok(userResponse);
14. }
```

ASP.NET Core Identity

User Logout

Program.cs:

```
1. [HttpPost("/api/auth/logout")]
2. public async Task<IActionResult> LogoutAsync()
3. {
4.     await _signInManager.SignOutAsync();
5.     return Ok();
6. }
```

Thank You