Digital Distribution Service

(Miron Catalin Andrei)

About

This project simulates the back-end structure for an online shop specialized for software products. To achieve this, I chose the code-first approach to create our SQL Server Database using C# and the Entity Framework Core.

```
namespace DigitalDistribution.Models.Database.Entities
{
    [Table("Addresses")]
    public class BillingAddressEntity:BaseEntity
    {
        public string ZipCode { get; set; }
        public string Country { get; set; }
        public string City { get; set; }
        public string Street { get; set; }
        public List<InvoiceEntity> Bills { get; set; }
        public int UserId { get; set; }
        [ForeignKey("UserId")]public UserEntity User { get; set; }
    }
}
```

To define our entities and the relationships between them we use the data annotations and the Fluent API available in EF Core.

```
 modelBuilder.Entity<UserEntity>()
             .HasOne(e => e.Address)
             .WithOne(e => e.User)
             .IsRequired(false)
             .OnDelete(DeleteBehavior.Cascade);
```

To allow the users to interact with our database, we created the HTTP services using the ASP.NET Web API framework.

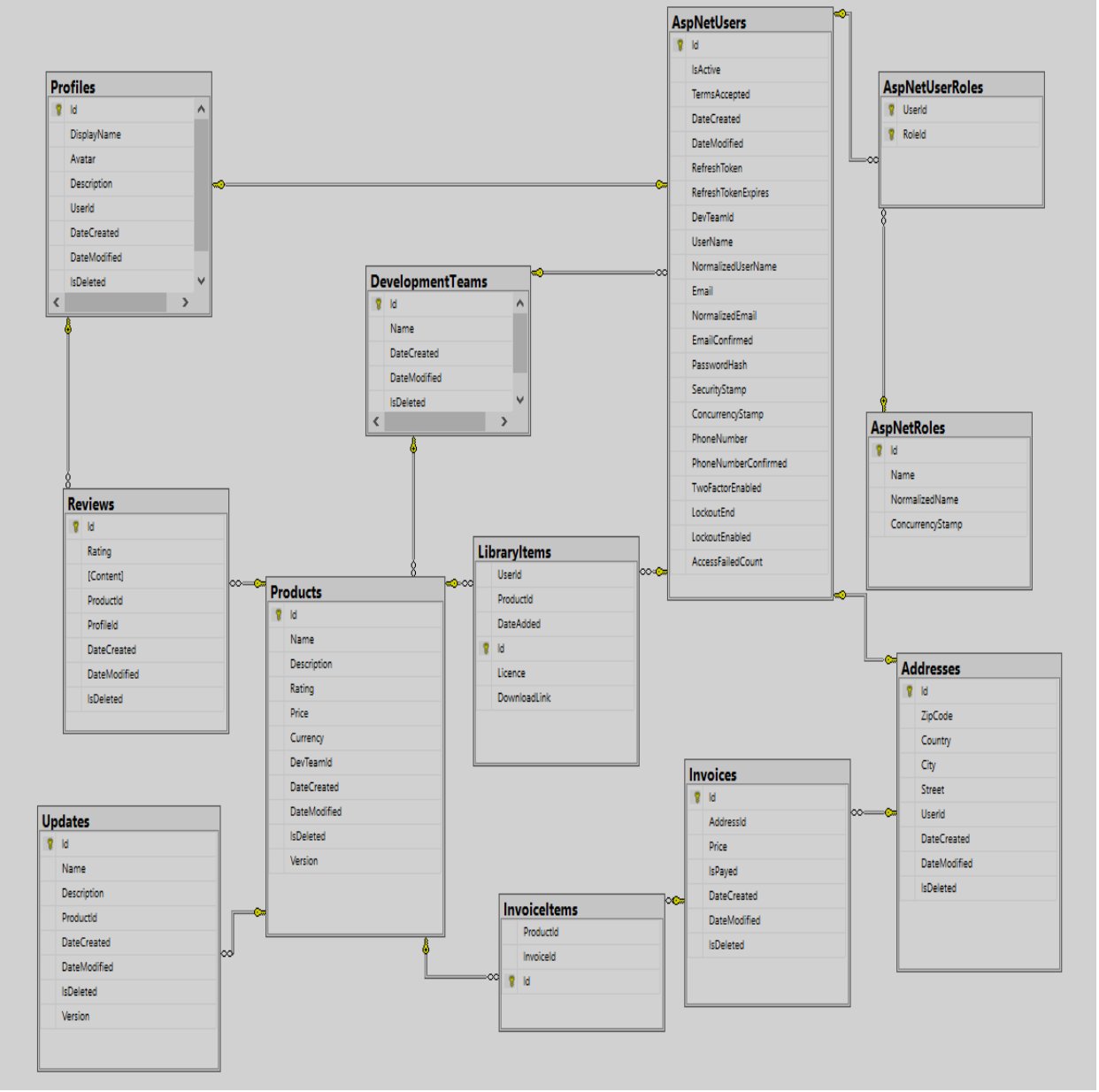
```
[Authorize]
[ApiController]
[Route("api/devTeams")]
public class DevelopmentTeamController:ControllerBase
{
        [HttpGet]
        public async Task<ObjectResult> GetAllDevelopmentTeams()
        {
                var result = await _developmentTeamService.Get()
                .Include(p=>p.Products)
                .ToListAsync();

                if (result is null)
                   throw new NotFoundException(StringConstants.NoDevTeams);

                return Ok(result);
        }
}
```

The business layer follows the Repository-Service pattern which means that it is split in 2 distinct layers.

We make use of polymorphism to make our code cleaner and easier to follow.

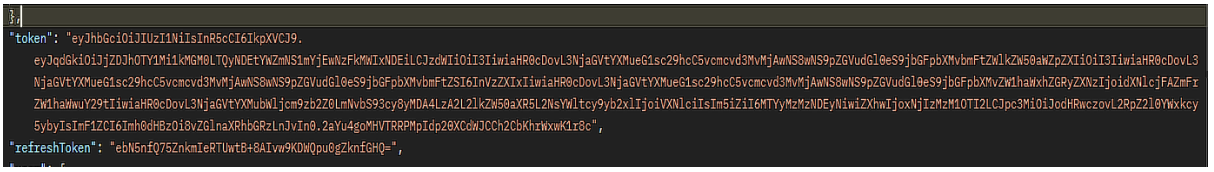
```
public class BaseRepository<T> where T : BaseEntity
    {
        protected readonly DigitalDistributionDbContext DbContext;
        protected readonly DbSet<T> Table;

        public BaseRepository(DigitalDistributionDbContext dbContext)
        {
            DbContext = dbContext;
            Table = DbContext.Set<T>();
        }
}
```

The DataBase



Functionality

When registering, the user will receive a role that limits the access to certain endpoints. When the user logs in, a token and a refresh token is generated to ensure security.



- The normal user can set up a profile, a billing address, can place orders and can leave reviews to the items that they bought.
- The developers have access to their own products that they published.
- The admins have total access to the products and development teams. That means that they can add new products/dev teams but they can also remove them or change them.

If an error pops up while we use the services, the error will be stored inside a logger.

Bibliography

- Overview of Entity Framework Core - EF Core
- ASP.NET documentation
- Get Started with ASP.NET Web API 2 (C#) - ASP.NET 4.x

Contact

- E-mail: cata.miron98@gmail.com
- GitLab: https://gitlab.com/cata.miron98