Part 2: Adding JWT Authentication

In this part, we will implement JWT-based authentication to secure our API. This includes user registration, login, and protecting product CRUD endpoints so only authenticated users can access them.

1. Install JWT NuGet Packages

Run the following commands to install the required JWT libraries:

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add package System.IdentityModel.Tokens.Jwt
```

2. Create the User Entity

- Add a `User` entity to the `Models` folder (`Models/User.cs`):

```
namespace ECommerceAPI.Models
{
        public class User
        {
        public int Id { get; set; }
        public string Username { get; set; }
        public string PasswordHash { get; set; }
        }
}
```

3. Update Database Context

- Add the `User` DbSet to `ECommerceDbContext`:

```
public DbSet<User> Users { get; set; }
```

Run migrations to update the database:

```
dotnet ef migrations add AddUserEntity
dotnet ef database update
```

4. Create Authentication Configuration

- Add a new class `AuthSettings.cs` in a folder called `Helpers`:

```
namespace ECommerceAPI.Helpers
{
	public class AuthSettings
	{
	public string Secret { get; set; }
	}
}
```

Add this configuration to `appsettings.json`:

```
"AuthSettings": {
  "Secret": "YourVeryStrongSecretKeyHere1234567890"
}
```

Register the `AuthSettings` in `Program.cs`:

```
using ECommerceAPI.Helpers;

builder.Services.Configure<AuthSettings>(builder.Configuration.GetSection("AuthSettings"));
```

5. Add Authentication Middleware

- In `Program.cs`, add JWT authentication configuration:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;

var authSettings = builder.Configuration.GetSection("AuthSettings").Get<AuthSettings>();

builder.Services.AddAuthentication(options =>
{
	options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
	options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
	options.TokenValidationParameters = new TokenValidationParameters
	{
	ValidateIssuerSigningKey = true,
	IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(authSettings.Secret)),
```

```
        ValidateIssuer = false,
        ValidateAudience = false
        };
});
```

Update middleware to use authentication and authorization:

```
app.UseAuthentication();
app.UseAuthorization();
```

6. Create a User Controller

- Add a `Controllers/UserController.cs`:

```
using ECommerceAPI.Data;
using ECommerceAPI.Helpers;
using ECommerceAPI.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace ECommerceAPI.Controllers
{
        [Route("api/[controller]")]
        [ApiController]
        public class UserController : ControllerBase
        {
        private readonly ECommerceDbContext _context;
        private readonly AuthSettings _authSettings;

        public UserController(ECommerceDbContext context, IOptions<AuthSettings>
authSettings)
        {
        _context = context;
        _authSettings = authSettings.Value;
        }

        // POST: api/User/register
        [HttpPost("register")]
        public async Task<IActionResult> Register(User user)
```

```
        {
        if (await _context.Users.AnyAsync(u => u.Username == user.Username))
        {
                return BadRequest("Username already exists.");
        }

        user.PasswordHash = BCrypt.Net.BCrypt.HashPassword(user.PasswordHash);
        _context.Users.Add(user);
        await _context.SaveChangesAsync();

        return Ok("User registered successfully.");
        }

        // POST: api/User/login
        [HttpPost("login")]
        public async Task<IActionResult> Login(User user)
        {
        var dbUser = await _context.Users.SingleOrDefaultAsync(u => u.Username ==
user.Username);

        if (dbUser == null || !BCrypt.Net.BCrypt.Verify(user.PasswordHash,
dbUser.PasswordHash))
        {
                return Unauthorized("Invalid username or password.");
        }

        var token = GenerateJwtToken(dbUser);
        return Ok(new { Token = token });
        }

        private string GenerateJwtToken(User user)
        {
        var tokenHandler = new JwtSecurityTokenHandler();
        var key = Encoding.UTF8.GetBytes(_authSettings.Secret);

        var tokenDescriptor = new SecurityTokenDescriptor
        {
                Subject = new ClaimsIdentity(new Claim[]
                {
                new Claim(ClaimTypes.Name, user.Id.ToString())
                }),
                Expires = DateTime.UtcNow.AddHours(1),
                SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature)
```

```
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
        }
        }
}
```

7. Secure the Products Controller

- Add the `[Authorize]` attribute to the `ProductsController` to restrict access:

```
using Microsoft.AspNetCore.Authorization;

[Authorize]
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
        // Existing code
}
```

8. Test the API

1. **Register a user**:
    - Endpoint: `POST /api/User/register`
    - Body:

```
{
  "username": "testuser",
  "passwordHash": "testpassword"
}
```

Login to get a token:

- Endpoint: `POST /api/User/login`
- Body:

```
{
  "username": "testuser",
  "passwordHash": "testpassword"
}
```

Response:

```
{
  "token": "your-jwt-token"
}
```

Access protected endpoints:

- Include the token in the `Authorization` header as `Bearer <token>` to access secured product endpoints.