REGISTER

1) DomainLayer-de Entity folderinde AppUser classi yaradiriq, prop-larimizi daxil edirik (mes: string Fullname) Identity netframework pack-ini yukleyirik ve classimiz IdentityUser classindan miras alir.

2) Repositorydeki Data folderinde AppDbContext-mize classimiz IdentityDbContext<AppUser> classindan miras aldiririq.

3)Pragram.cs-e asagidaki code-u elave edirik:

```
builder.Services.AddIdentity<AppUser, IdentityRole>()
.AddEntityFrameworkStores<AppDbContext>();
```

4)Migration edirik ve sql-de table-lerimiz yaranir. (add-migration, update-database)

5) ServiceLayer-de Services folderinin icindeki Interfaces folderinde IAccountService interface-ni yaradiriq.

6) ServiceLayer-de DTO folderinin icinde Account folderi yaradiriq: Bu folderin icinde 2 class olur:

1-LoginDto.cs: bu classin icinde Email ve Password proplari yaradiriq.

2-RegisterDto.cs: bu classin icinde Username, Fullname, Email, Password proplari olur. (Elave proplar da qeyd ede bilerik)

7) IAccountService-de yaratdigimiz Dto-lari cagiririq: Task LoginAsync(LoginDto model), Task RegisterAsync (RegisterDto model)

8) Mapping folderimizde MappingProfile classimizda datalarimizi mapping edirik:

```
CreateMap<RegisterDto, AppUser>().ReverseMap();
```

9) Service folderimizde AccountService classi yaradiriq, classimiz IAccountService interface-den miras alir. Interface-i implement edirik.

Davaminda bu class-a asagidakilari elave edirik:

```
    private readonly UserManager<AppUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;
    private readonly IMapper _mapper;
    private readonly IConfiguration _configuration

    public AccountService(UserManager<AppUser> userManager, RoleManager<IdentityRole> roleManager, IMapper mapper, IConfiguration configuration)
    {
        _userManager = userManager;
        _roleManager = roleManager;
        _mapper = mapper;
        _configuration = configuration;
    }
```

10) AccountService-de implement etdiyimiz RegisterAsync metodumuzun icini doldururuq:

```
var user = _mapper.Map<AppUser>(model);
```

```
IdentityResult result = await _userManager.CreateAsync(user,model.Password);
```

11) Login ve Register prosseslerinde butun response-lar ucun Account folderinde ApiResponse class-I yaradiriq, bu classimizin iki prop-u olur:

```
public List<string>? Errors { get; set; }
public string? StatusMessage { get; set; }
```

12) Davaminda AccountService classimiza daxil olub respone-u istifade etmek ucun RegisterAsync metodumuzun davamina elave edirik:

```
if (!result.Succeeded)
        {
            ApiResponse response = new()
            {
                Errors = result.Errors.Select(m => m.Description).ToList(),
                StatusMessage = "Failed"
            };

            return response;

        }

        return new ApiResponse { Errors = null, StatusMessage= "Success" };
```

12) Register metodumuzun bu tipden bize qaytarmasi ucun IAccountService interface-mizde ve AccountService classimizda bunu metoda qeyd edirik:

```
Task<ApiResponse> RegisterAsync(RegisterDto model);
```

```
public async Task<ApiResponse> RegisterAsync(RegisterDto model)
```

13) App-de Program.cs-de AddScope code-larimizi qeyd edirik edirik:

```
builder.Services.AddScoped<IAccountService, AccountService>();
```

14) App-de Controller folderinde AccountController yaradiriq, bu controller AppController-den miras alir. Son olaraq bu controller-in icine asagidakilari qeyd edirik:

```
  private readonly IAccountService _accountService;

      public AccountController(IAccountService accountService)
      {
         _accountService = accountService;
      }

      [HttpPost]
      public async Task<IActionResult> Register([FromBody]RegisterDto user)
      {
         return Ok (await _accountService.RegisterAsync(user));
      }
   }
```

JWT KONFIQURASIYALARINI QURMAQ

1) Asagidaki kodu appsettings.json-a elave edirik:

```
{
  "Jwt": {
  "Key": "a-very-long-radonmly-generated-secret-key-that-cannot-be-guessed",
  "Issuer": "https //localhost:7243:44394",
  "Auidience": "https://localhost:7243
  "ExpireDays": 30
},
```

```
"Logging": {
 "LogLevel": {
  "Default": "Information",
  "Microsoft.AspNetCore": "Warning"
```

```
  }
 },
 "AllowedHosts": "*"
   }
```

2) Program.cs -e asagidaki kodlari elave edirik:

```
JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();
builder.Services
   .AddAuthentication(options =>
   {
      options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
      options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
      options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
   })
   .AddJwtBearer(cfg =>
   {
      cfg.RequireHttpsMetadata = false;
      cfg.SaveToken = true;
      cfg.TokenValidationParameters = new TokenValidationParameters
      {
         ValidIssuer = builder.Configuration["Jwt:Issuer"],
         ValidAudience = builder.Configuration["Jwt:Auidience"],
         IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JwtKey"])),
         ClockSkew = TimeSpan.Zero
      };
   });
```

3) ServiceLayer-de Services folderindeki AccountService classina asagidaki “token yaratma” metodunu elave edirik:

```
private string GenerateJwtToken(string username, List<string> roles)
      {
         var claims = new List<Claim>
      {
         new Claim(JwtRegisteredClaimNames.Sub, username),
         new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
         new Claim(ClaimTypes.NameIdentifier, username)
      };

         roles.ForEach(role =>
         {
```

```
            claims.Add(new Claim(ClaimTypes.Role, role));
        });

        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JwtKey"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
        var expires =
DateTime.Now.AddDays(Convert.ToDouble(_configuration["JwtExpireDays"]));

        var token = new JwtSecurityToken(
            _configuration["Jwt:Issuer"],
            _configuration["Jwt:Auidience"],
            claims,
            expires: expires,
            signingCredentials: creds
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
```

LOGIN

1) IAccountService interface-de LoginAsync-ni asagidaki kodla evezleyirik:

```
Task<string?> LoginAsync(LoginDto model);
```

2)AccountService classimizda interface-i implement edib LoginAsync metodumuzun icini doldururuq:

```
public async Task<string?> LoginAsync(LoginDto model)
    {
        var dbUser = await _userManager.FindByEmailAsync(model.Email);
```

```
        if(!await _userManager.CheckPasswordAsync(dbUser, model.Password))
            return null;

        var roles = await _userManager.GetRolesAsync(dbUser);

        return GenerateJwtToken(dbUser.UserName, (List<string>)roles);
    }
```

3) AccountController class-imizda Login metodumuzu yaziriq:

```
[HttpPost]
    public async Task<IActionResult> Login([FromBody] LoginDto user)
    {
        return Ok(await _accountService.LoginAsync(user));
    }
```

ROLLAR YARATMAQ

1) DTO folderinin icindeki Account folderinde RoleDto classi yaradiriq ve classa role prop-u elave edirik.

2) IAccountService interface-e asagidaki kodu elave edirik:

```
Task CreateRoleAsync(RoleDto model);
```

3) AccountService classimizda interface-I implement edib yaranan CreateRole metodumuzun icini doldururuq:

```
public async Task CreateRoleAsync(RoleDto model)
    {
        await _roleManager.CreateAsync(new IdentityRole { Name = model.Role });
    }
```

4) AccountControllerimizde yeni CreateRole metodu yaradiriq:

```
[HttpPost]
        public async Task<IActionResult> CreateRole([FromBody] RoleDto role)
        {
            await _accountService.CreateRoleAsync(role);
            return Ok();
        }
```

5) AccountService classina daxil oluruq RegisterAsync metoduna asagida qeyd etdiyimiz yeni kodlari elave edirik:

```
public async Task<ApiResponse> RegisterAsync(RegisterDto model)
        {
            var user = _mapper.Map<AppUser>(model);

            IdentityResult result = await _userManager.CreateAsync(user,model.Password);

            if (!result.Succeeded)
            {
                ApiResponse response = new()
                {
                    Errors = result.Errors.Select(m => m.Description).ToList(),
                    StatusMessage = "Failed"
                };

                return response;

            }
            var dbUser = await _userManager.FindByEmailAsync(model.Email);

            await _userManager.AddToRoleAsync(dbUser,"Admin");

            return new ApiResponse { Errors = null, StatusMessage= "Success" };
        }
```

6) Son olaraq hansi rolun hansi metodlari istifade edeceyini teyin etmek ucun (mes: BookControllerde Create metodu) controllerde o metodun uzerine asagidaki kod yazilir:

`[Authorize(Roles = “Admin”)]` ve ya `[Authorize(Roles = “Member”)]`

(Bu funksiyani istifade etmek ucun program.cs-de `app.UseAuthentication` qeyd etmeliyik.)