Uygulama Mimarisi kalıpları : Mikroservis Mimarisi

Hazırlayan : **Oğuzhan Seleker**
Öğrenci No : **090170325**

Teslim Tarihi : **15.01.2021**

Ders : **MAT 4901**
Danışman : **PROF. DR. FÜSUN ZENGİN**

İçindekiler Tablosu

Şekiller Tablosu

1. Tasarımın Tanımı ve Amacı

Bu çalışmanın amacı, günümüzdeki büyük teknoloji firmaları tarafından kullanılan ufak takımlar halinde ve birbirlerinden olabildiğince bağımsız çalışmaya olanak sağlayan bir uygulama tasarım mimarisi olan mikroservis mimarisi [1] ile öğrenciler ve öğretim görevlileri tarafından kullanılabilecek ders yönetim ve takip uygulaması tasarlamaktır.

2. Tasarımın Kapsamı ve Kullanım Alanları

Bu tasarımda, uygulama tasarım mimarileri araştırılacak ve mikroservis mimarisinin ne olduğu, niçin teknoloji firmaları tarafından tercih edildiği, avantajları ve dezavantajları ele alınacak. Mikroservis mimarisinin diğer tasarım mimarileri arasındaki farkları, avantatajları ve dezavantajları ele alınacaktır. Mikroservis mimarisinin uygulama geliştirme sürecinde sağladığı kolaylıklar ve zorluklar araştırılacaktır.

Çalışmaların çıktısı olarak, Ders Yönetim Sistemi projesi yapılması planlanmaktadır. Proje oluşturulurken C# programlama dili kullanılacaktır. Projede on adet, her birinin farklı görevleri olan .Net 5.0 ile mikroservis, üyelik sistemi ve yetkilendirme için açık kaynaklı kodlu IdentityService4 [2] entegre edilmiş .Net Core 3.1 servis, bir adet kullanıcılara sunulacak Asp.Net 5 MVC [3] ile web uygulaması, web uygulaması ile servisleri haberleştirmek için Ocelot Kütüphanesi [4]ile API Gateway yazılacaktır.

Mikroservis mimarisi ile oluşturulması planlanan Ders Yönetim Sistemi projesinde online ortamda dersin yönetilmesi sağlanacaktır. Öğretim görevlileri bulunduğu dönemde açtığı dersi oluşturabilecek ve dersi alan öğrencileri ekleyebileyebilecektir. Ders oluşturulduktan sonra içerisinde bir çok fonksiyon barındıracaktır. Bu fonksiyonlar;

- Derse birden fazla öğretim görevlisi ve yardımcı öğretim görevlileri eklenebilecektir.
- Dersin bilgilerine erişilebilecektir.
- Mesaj kutusu sayesinde dersteki herkes birbirleri ile anlık iletişim kurabilecektir.
- Duyuru bölümünde öğretim görevlileri ve yardımcı öğretim görevlileri duyuru oluşturabilecektir.
- Anket modülü sayesinde öğrencilere bir konuda fikirleri alınıp hayata geçirilebilecektir.
- Ödev modülü sayesinde online ortamda ödev verilebilecektir.
- Yoklama bölümünde derse katılan katılmayan öğrencilerin bilgisi kayıt altına alınabilecektir.
- İlgili dosyalar modülünde ders ile ilgili dosyalar, bağlantılar eklenebilecektir.
- Mini quiz modülünde haftalık mini quizler gerçekleştirilip notlandırması yapılabilecektir.
- Üyelik ve rol bazlı yetkilendirme sayesinde bilgilerin güvenliği sağlanabilecektir.

Tasarımın dağıtım stratejisinde son zamanlarda popülerleşmiş olan Docker paketleme ve dağıtma servisi kullanılacaktır. Açık kaynak kodlu olan Docker bir "container" teknolojisidir. Birden fazla görüntüyü birbirinden bağımsız şekilde sanallaştırmayı sağlayan bir teknolojidir. Bundan dolayı da mikroservis mimarisinde Docker teknolojisinin kullanımı oldukça avantajlıdır.

3. Yapılan Çalışmalar

Uygulama mimarisi kalıpları; monolitik mimari avantajları ve dezavantajları, N-Tier Architecture mimari ile servis geliştirilmesi, mikroservis mimarisinin temelleri, teorik çalışmalarının yapılması, mikroservisinin diğer mimarilere göre farkları öne çıkan özellikleri, mikroservis mimarisine en uygun tasarım desenlerinin (Design Patterns) araştırılması, Repository Design Pattern(Repository tasarım deseni) [5] araştırırılması ve uygulanması, Command Query Responsibility Segregation Design Pattern'inin(CQRS - komut ve sorgu sorumluluklarının ayrışması) [6] araştırılması ve örneklerle uygulanması, web servislerinin çalışma mantığının araştırılması, en performanslı ve en iyi pratiklerinin öğrenilmesi, .Net Core platformunda geliştirilmiş üyelik sistemlerinin araştırılması, en güvenli yöntemler ile geliştirilmiş açık kaynak kodlu projelerin araştırılması, OpenId ve Oauth 2.0 yöntemlerinin araştırılması, rol bazlı yetkilendirme kurallarının araştırılıp araştırılması, yapılacak projeyi mikroservis mimarisine göre planlamak, kullanılacak veritabanı araştırılması, Docker temellerinin öğrenilmesidir.

3.1 Ders Yönetim Sistemi Projesinde Yapılması Planlananlar ve Yapılanlar

3.1.1 Bu Proje Kapsamında Yapılması Planlananlar

- Mikroservisler;
    - IdentityServer Mikroservisi : .Net Core platformunda 3.1 versiyonunu kullanan ve içerisinde açık kaynak kodlu IdentityServer4 kütüphanesini barındıran, üyelik sisteminden sorumlu, kişinin bilgilerini güvenli bir şekilde tutan ve paylaşan bir servistir. Veritabanı olarak Microsoft SQL Server kullanmaktadır.
    - Lesson Mikroservisi: .Net Core platformunda 5.0 versiyonunu kullanan ve ders ile ilgili işlemleri sunan bir servistir. Dersin oluşturulmasını, düzenlenmesini, silinmesini ve depolanmasından sorumludur. Veritabanı olarak PostgreSql kullanmaktadır.
    - Homework Mikroservisi : .Net Core platformuna 5.0 versiyonunu kullanan ve ödevlerin bilgisini, ödev dosyasının kaynağını tutan, öğrencilerin yüklediği ödev dosyalarını tutan ve bunu sunan bir servistir. Veritabanı olarak PostgreSql kullanmaktadır.
    - Attendance Mikroservisi : .Net Core platformuna 5.0 versiyonunu kullanılarak yazılacaktır. Derslerin hafta hafta öğrenci katılımlarını tutucak ve sunacak olan servistir. Veritabanı olarak PostgreSql kullanılacaktır.

- Message Mikroservisi : .Net Core platformuna 5.0 versiyonunu kullanılarak yazılacaktır. Öğrencilerin Öğretim görevlileriyle ve yardımcı öğretim görevlileriyle mesajlaşmayı sağlayan servistir. Veriler PostgreSql veritabanında saklanacaktır.
- Survey Mikroservisi : .Net Core platformunda 5.0 versiyonu kullanarak, Öğretim görevlilerinin öğrencilere uygulayacağı anketlerin operasyonlarının gerçekleştirileceği servistir. Veritabanı olarak PostgreSql kullanılacaktır.
- Quiz Mikroservisi : Quiz sisteminden sorumlu olan, öğretim görevlilerinin eklediği soruları öğrenciler tarafından anlık olarak cevaplanacağı ve bu işlemlerin saklanacağı servistir. .Net Core 5.0 kullanılarak yazılacaktır. Veritabanı olarak PostgreSql kullanılacaktır.
- Grading Mikroservisi : Homework mikroservisinde ve quiz mikroservisinde gerçekleşen işlemlerin öğretim görevlileri tarafından notlandırma işlemlerinin yapılacağı servistir. .Net Core 5.0 ile yazılacaktır. Veritabanı olarak PostgreSql kullanılacaktır.

- Ocelot ile Gateway Servisi : Mikroservis mimarisinde en çok kullanılan açık kaynak kodlu Mikroservisler arasında yönlendirme ve haberleşmeyi sağlayan Ocelot kütüphanesi araştırılacaktır ve projeye entegre edilecektir.
- Asp.Net 5 MVC Web Uygulaması : Asp.Net 5 platformunda MVC(Model-View-Controller) mimari desen kullanılarak servislerin kullanılacağı web uygulaması yapılacaktır.

3.1.2 Bu Proje Kapsamında Birinci Dönem Yapılanlar

- Projenin mikroservis mimarisinde nasıl planlanması gerektiği araştırıldı.
- 11 adet mikroservis geliştirilmesi amaçlandı.
- Geliştirilecek mikroservislerin veritabanları ve çalıştırılacak ortamları planlandı.
- Proje kodlarını barındıran çözüm dosyası eklendi.
- Lesson mikroservisi Repository tasarım kalıbı ile oluşturuldu. Database bağlantısı yapıldı. Testleri yapıldı.(Bkz. *Şekil 5*)
- IdentityServer mikroservisi kuruldu. Açık kaynak kodlu IdentityServer4 kütüphanesi entegre edildi. Test Kullanıcıları oluşturuldu. Üyelik işlemleri ve giriş işlemleri tamamlandı.(Bkz. *Şekil 3*)
- IdentityServer servisinin Sql Server bağlantısı gerçekleştirildi. (Bkz. *Şekil 6*)
- IdentityServer mikroservisine istemci(client) olarak Asp.Net Core 5.0 MVC Web uygulaması eklendi.(Bkz. *Şekil 4*)
- Eklenen web uygulaması ile IdentityServer'dan token alma işlemleri gerçekleştirildi.(Bkz. *Şekil 1*)
- Yazılan Mikroservisler ve web uygulaması IdentityServer tarafından koruma altına alındı.(Bkz. *Şekil 2*)

- Web uygulamanın mikroservislere erişebilmesi için açık kaynak kodlu Ocelot kütüphanesi kullanılarak API Gateway eklendi.
- Mikroservislerde ve İdentityServer'da kullanılan veritabanları Docker'da ayağa kaldırıldı.(Bkz. *Şekil 7*)

3.2 Uygulamadan Görüntüler

```
namespace DYS.WebClient.Controllers
{

    1 reference | Oğuzhan Seleker, 7 days ago | 1 author, 1 change
    public class AuthController : Controller
    {
        private readonly IIdentityService _identityService;

        0 references | Oğuzhan Seleker, 7 days ago | 1 author, 1 change
        public AuthController(IIdentityService identityService)
        {
            _identityService = identityService;
        }
        0 references | Oğuzhan Seleker, 7 days ago | 1 author, 1 change
        public IActionResult SignIn()
        {
            return View();
        }
        [HttpPost]
        0 references | Oğuzhan Seleker, 7 days ago | 1 author, 1 change
        public async Task<IActionResult> SignIn(SigninInput signinInput)
        {
            if (!ModelState.IsValid) return View();

            var response = await _identityService.SignIn(signinInput);
            if (!response.Success)
            {
                ModelState.AddModelError("error",response.ErrorMessage);
                return View();
            }

            return RedirectToAction(nameof(Index), "Home");
        }

        0 references | 0 changes | 0 authors, 0 changes
        public async Task<IActionResult> LogOut()
        {
            await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
            await _identityService.RevokeRefreshToken();
            return RedirectToAction(nameof(HomeController.Index), "Home");

        }
    }
}
```

Şekil 1: Web uygulamasında Üye Giriş ve Çıkış işlemleri

```
2 references | Oğuzhan Seleker, 7 days ago | 1 author, 3 changes
public class Startup
{
    0 references | Oğuzhan Seleker, 14 days ago | 1 author, 1 change
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    4 references | Oğuzhan Seleker, 14 days ago | 1 author, 1 change
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    0 references | Oğuzhan Seleker, 7 days ago | 1 author, 3 changes
    public void ConfigureServices(IServiceCollection services)
    {
        services.Configure<ClientSettings>(Configuration.GetSection("ClientSettings"));
        services.Configure<ServiceApiSettings>(Configuration.GetSection("ServiceApiSettings"));
        services.AddScoped<ResourceOwnerPasswordTokenHandler>();
        var serviceApiSettings = Configuration.GetSection("ServiceApSettings").Get<ServiceApiSettings>();
        services.AddHttpContextAccessor();
        services.AddHttpClient<IIdentityService, IdentityService>();
        services.AddHttpClient<IUserService, UserService>(opt =>
        {
            opt.BaseAddress = new Uri(serviceApiSettings.IdentityBaseUri);
        }).AddHttpMessageHandler<ResourceOwnerPasswordTokenHandler>();

        services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, opt =>
        {
            opt.LoginPath = "/Auth/SignIn";
            opt.ExpireTimeSpan = TimeSpan.FromDays(60);
            opt.SlidingExpiration = true;
            opt.Cookie.Name = "dyswebcookie";
            opt.AccessDeniedPath = "/home/AccessDenied";
        });
        services.AddControllersWithViews();
    }
```

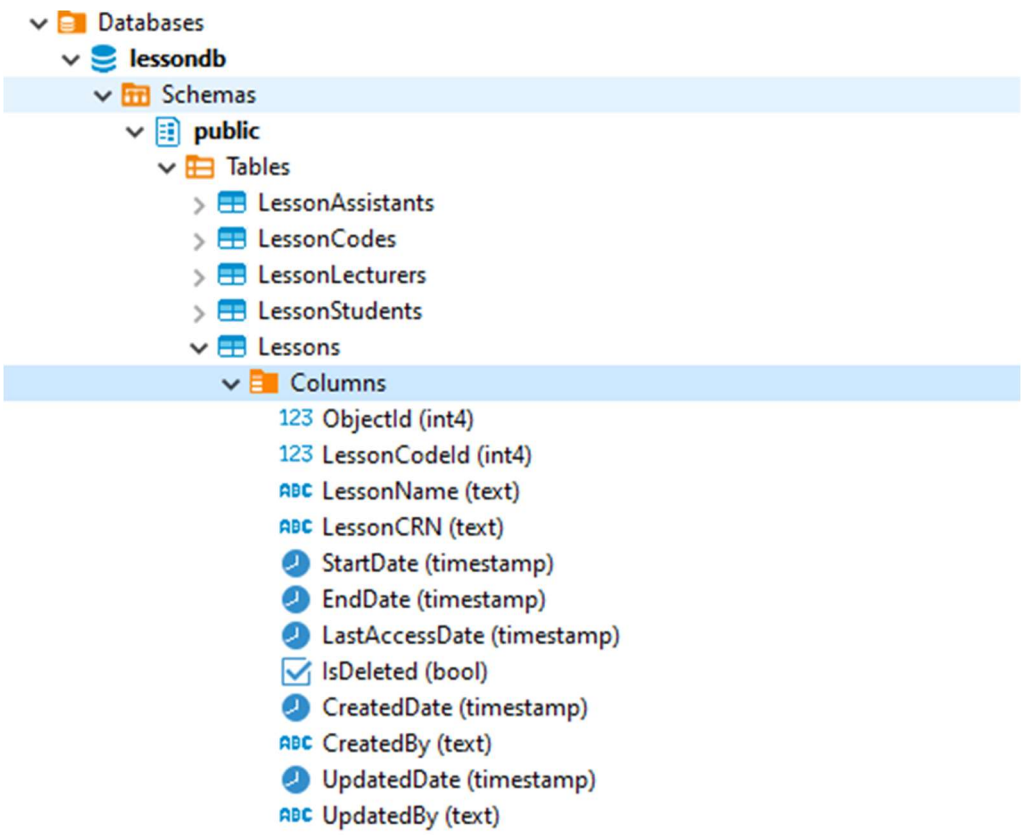
Şekil 2: Web Uygulamasında servis konfigürasyonlarının yapılması

```
[ApiController]
1 reference | Oğuzhan Seleker, 9 days ago | 1 author, 1 change
public class UserController : CustomBaseController
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;

    0 references | Oğuzhan Seleker, 5 days ago | 1 author, 1 change
    public UserController(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager)
    {
        _userManager = userManager;
        _roleManager = roleManager;
    }

    [HttpPost]
    0 references | Oğuzhan Seleker, 5 days ago | 1 author, 1 change
    public async Task<IActionResult> SignUp(SignUpDto signupDto)
    {
        var user = new ApplicationUser
        {
            UserName = signupDto.UserName,
            Email = signupDto.Email,
            FirstName = signupDto.FirstName,
            LastName = signupDto.LastName
        };
        var existRole = await _roleManager.RoleExistsAsync(signupDto.RoleName);
        if (!existRole) return CreateActionResultInstance(OperationResult<NoContent>.CreateFailure("Role Bulunamadı.", SharedLibrary.ResponseDtos.StatusCode.NotFound));
        var result = await _userManager.CreateAsync(user, signupDto.Password);
        if (!result.Succeeded)
        {
            return CreateActionResultInstance(OperationResult<NoContent>.CreateFailure("Kayıt olurken bir hata ile karşılaşıldı. " + result.Errors.FirstOrDefault().Description,
              SharedLibrary.ResponseDtos.StatusCode.Error));
        }
        var resultRole = await _userManager.AddToRoleAsync(user, signupDto.RoleName);
        if (!resultRole.Succeeded)
        {
            return CreateActionResultInstance(OperationResult<NoContent>.CreateFailure("Rol eklenirken bir hata ile karşılaşıldı. Mesaj : " + resultRole.Errors.FirstOrDefault
              ().Description, SharedLibrary.ResponseDtos.StatusCode.Error));
        }

        return CreateActionResultInstance(OperationResult<NoContent>.CreatedSuccessResult());
    }

    [HttpGet]
    0 references | Oğuzhan Seleker, 5 days ago | 1 author, 1 change
    public async Task<IActionResult> GetUser()
    {
        var userIdClaim = User.Claims.FirstOrDefault(x => x.Type == JwtRegisteredClaimNames.Sub);
        if (userIdClaim == null) return CreateActionResultInstance(OperationResult<NoContent>.CreateFailure("UserClaim Not
          Found",SharedLibrary.ResponseDtos.StatusCode.BadRequest));

        var user = await _userManager.FindByIdAsync(userIdClaim.Value);
        if (user == null) return CreateActionResultInstance(OperationResult<NoContent>.CreateFailure("UserClaim Not Found", SharedLibrary.ResponseDtos.StatusCode.BadRequest));
        var userRole = await _userManager.GetRolesAsync(user);
        return CreateActionResultInstance(OperationResult<GetUserDto>.OkSuccessResult(new GetUserDto { Id = user.Id, UserName = user.UserName, Email = user.Email, FirstName =
          user.FirstName, LastName = user.LastName, Roles = userRole }));
    }
}
```

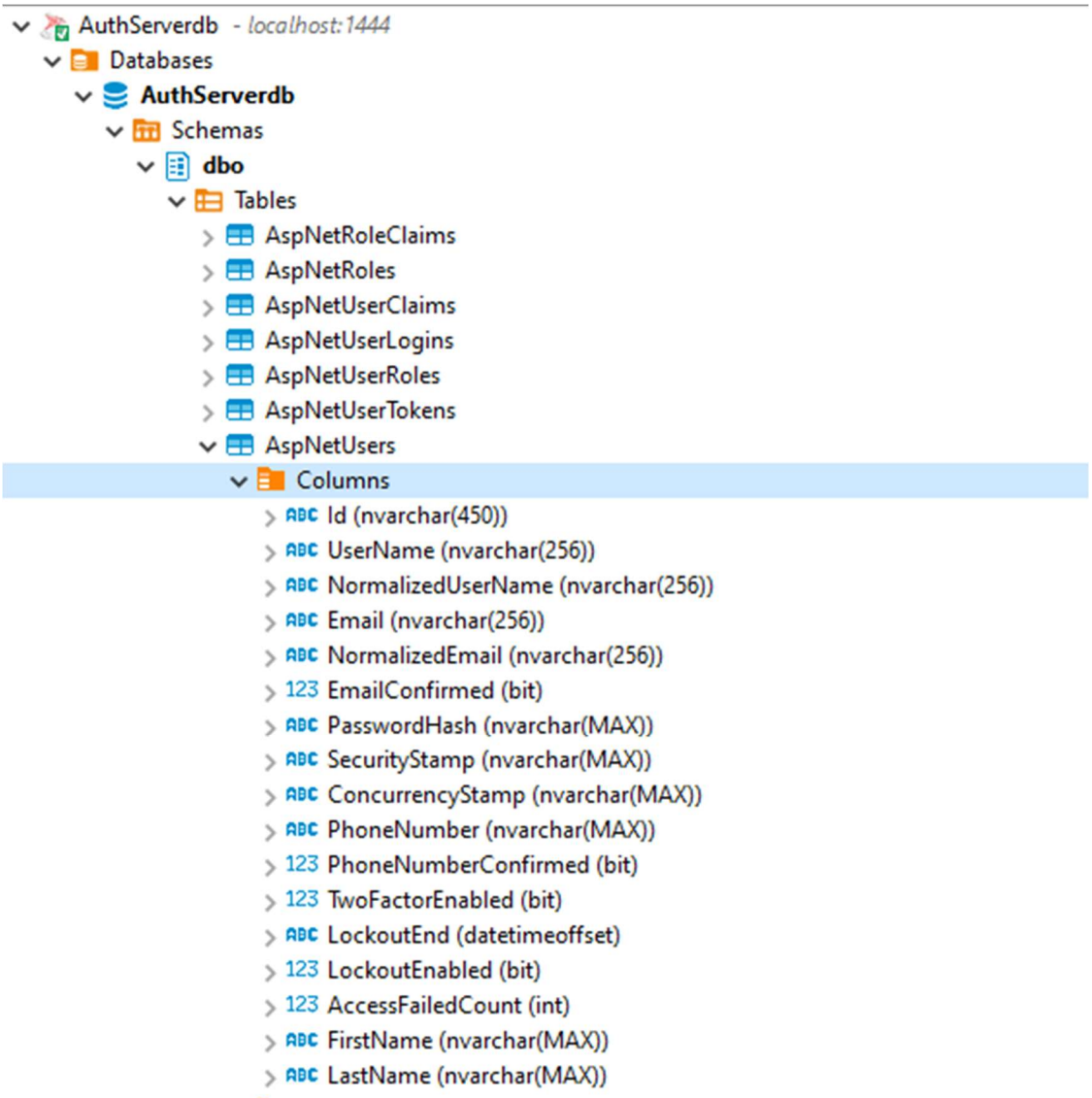
Şekil 3 : IdentityServer Servisinde SignUp ve GetUser Endpointleri

```
4 references | Oğuzhan Seleker, 7 days ago | 1 author, 2 changes
public static class Config
{
    1 reference | Oğuzhan Seleker, 7 days ago | 1 author, 2 changes
    public static IEnumerable<ApiResource> ApiResources => new ApiResource[]
            {
        new ApiResource("resource_lesson"){Scopes={"lesson_fullpermission"}},

        new ApiResource(IdentityServerConstants.LocalApi.ScopeName)
            };

    1 reference | Oğuzhan Seleker, 7 days ago | 1 author, 2 changes
    public static IEnumerable<IdentityResource> IdentityResources =>
            new IdentityResource[]
            {
                new IdentityResources.Email(),
                new IdentityResources.OpenId(),
                new IdentityResources.Profile(),
                new IdentityResource(){Name = "Roles",DisplayName="Roles",Description="Kullanıcı rolleri",UserClaims= new []{ "role"} }
            };

    1 reference | Oğuzhan Seleker, 9 days ago | 1 author, 1 change
    public static IEnumerable<ApiScope> ApiScopes =>
        new ApiScope[]
        {
            new ApiScope("lesson_fullpermission","Lesson API için full erişim"),
            //new ApiScope("gateway_fullpermission","Gateway API için full erişim"),
            new ApiScope(IdentityServerConstants.LocalApi.ScopeName),

        };

    1 reference | Oğuzhan Seleker, 7 days ago | 1 author, 2 changes
    public static IEnumerable<Client> Clients =>
        new Client[]
        {
            new Client
            {
                ClientName="DYS Web Client",
                ClientId = "WebMvcClientForUser",
                AllowOfflineAccess = true,
                ClientSecrets={new Secret("secret".Sha256())},
                AllowedGrantTypes = GrantTypes.ResourceOwnerPassword,
                AllowedScopes={ "lesson_fullpermission", IdentityServerConstants.StandardScopes.Email, IdentityServerConstants.StandardScopes.OpenId,
                  IdentityServerConstants.StandardScopes.Profile, IdentityServerConstants.StandardScopes.OfflineAccess,"Roles", IdentityServerConstants.LocalApi.ScopeName},
                AccessTokenLifetime = 1*60*60, // 1 saat
                RefreshTokenExpiration = TokenExpiration.Absolute,
                AbsoluteRefreshTokenLifetime = (int)(DateTime.Now.AddDays(60)- DateTime.Now).TotalSeconds,
                RefreshTokenUsage = TokenUsage.ReUse
            }
        };
}
```

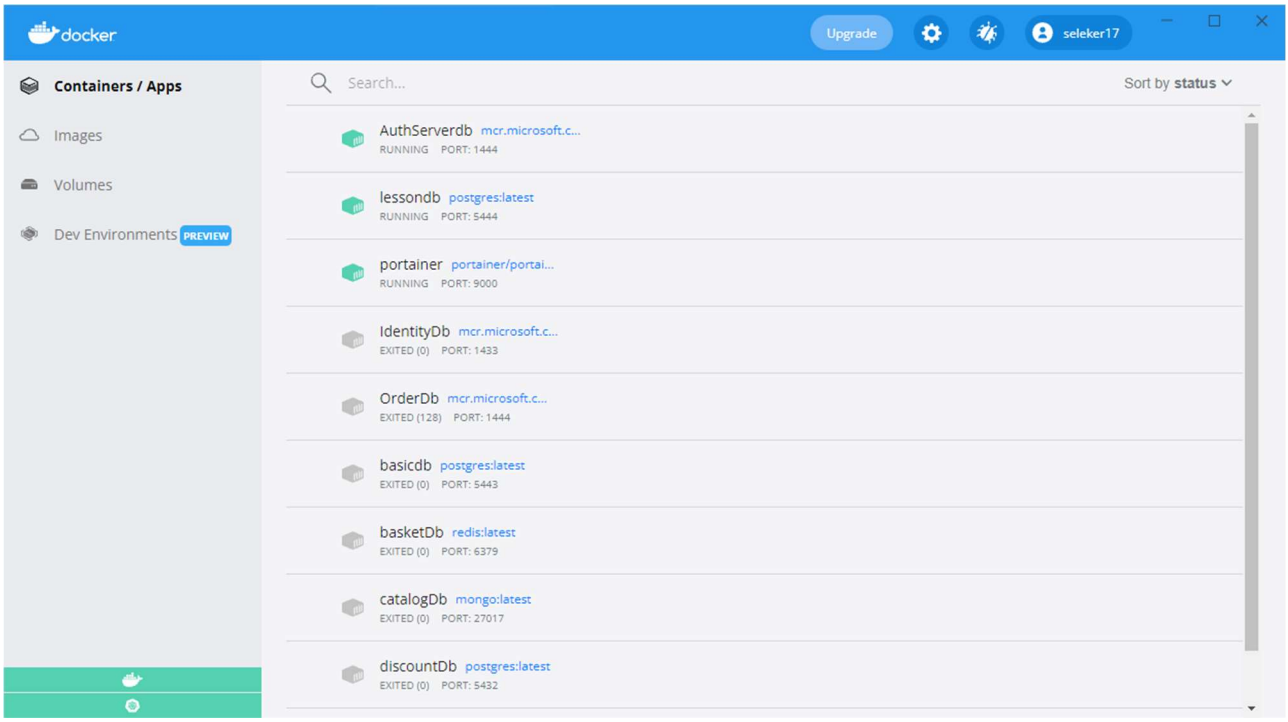
Şekil 4: IdentityServer Config dosyası



Şekil 5: Lesson mikroservisinin Veritabını



Şekil 6: Kullanıcı bilgilerinin tutulduğu veritabanı



Şekil 7: Veritabanlarının Docker ortamında ayağa kaldırılması

4. Kaynaklar

[1] https://microservices.io/patterns/microservices.html

[2] https://identityserver4.readthedocs.io/en/latest/quickstarts/0_overview.html

[3] https://docs.microsoft.com/tr-tr/aspnet/mvc/overview/getting-started/introduction/getting-started

[4] https://docs.microsoft.com/tr-tr/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot

[5] https://docs.microsoft.com/tr-tr/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

[6] https://docs.microsoft.com/tr-tr/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/apply-simplified-microservice-cqrs-ddd-patterns