FreeBirds API Detaylı Teknik Dokümantasyonu

Bölüm 1: Proje Yapısı ve Temel Konfigürasyon

1.1 Proje Mimarisi ve Katmanlı Yapı

1.1.1 Proje Organizasyonu

```
FreeBirds/
├── Controllers/         # API endpoint'leri - HTTP
isteklerini işleyen sınıflar
├── Services/           # İş mantığı servisleri -
Veritabanı işlemleri ve iş kuralları
├── Models/            # Veri modelleri - Veritabanı
tablolarını temsil eden sınıflar
├── Data/              # Veritabanı işlemleri - DbContext
ve migration'lar
├── DTOs/              # Data Transfer Objects - API
istek/yanıt modelleri
├── Migrations/        # Entity Framework migration'ları -
Veritabanı şema değişiklikleri
└── Properties/        # Proje özellikleri - Launch
settings ve diğer konfigürasyonlar
```

Açıklama: Bu yapı, temiz kod prensiplerine uygun bir katmanlı mimariyi temsil eder. Her katman kendi sorumluluğuna sahiptir ve diğer katmanlarla minimum bağımlılık içerir.

1.2 Konfigürasyon Dosyaları ve Yapılandırma

1.2.1 appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=FreeBirds.db"
  },
  "JwtSettings": {
    "SecretKey":
"Ub21qXRgJ/Fkoh6VEACotF/21MssZ1ziZdDWcYuO6Ss=",
```

```
    "Issuer": "FreeBirds",
    "Audience": "FreeBirdsApi",
    "AccessTokenExpirationMinutes": 60,
    "RefreshTokenExpirationDays": 7
  },
  "EmailSettings": {
    "SmtpServer": "smtp.gmail.com",
    "SmtpPort": 587,
    "SmtpUsername": "your-email@gmail.com",
    "SmtpPassword": "your-app-password",
    "FromEmail": "your-email@gmail.com",
    "FromName": "FreeBirds"
  },
  "AdminUser": {
    "Username": "admin",
    "Password": "Admin123!",
    "Email": "admin@freebirds.com",
    "FirstName": "Admin",
    "LastName": "User",
    "PhoneNumber": "1234567890"
  }
}
```

Açıklama:

- `ConnectionStrings`: Veritabanı bağlantı bilgilerini içerir
- `JwtSettings`: JWT token oluşturma ve doğrulama ayarları
- `EmailSettings`: E-posta gönderimi için SMTP sunucu ayarları
- `AdminUser`: Sistem yöneticisi kullanıcısının bilgileri

1.2.2 Konfigürasyon Sınıfları

```
// JwtSettings.cs
public class JwtSettings
{
    public string SecretKey { get; set; }        // JWT
token'ları imzalamak için kullanılan gizli anahtar
    public string Issuer { get; set; }           //
Token'ı yayınlayan taraf
    public string Audience { get; set; }         //
Token'ın hedef kitlesi
    public int AccessTokenExpirationMinutes { get; set; }
// Access token'ın geçerlilik süresi
    public int RefreshTokenExpirationDays { get; set; }
// Refresh token'ın geçerlilik süresi
}

// EmailSettings.cs
```

```
public class EmailSettings
{
    public string SmtpServer { get; set; }       // SMTP
sunucu adresi
    public int SmtpPort { get; set; }            // SMTP
port numarası
    public string SmtpUsername { get; set; }     // SMTP
kullanıcı adı
    public string SmtpPassword { get; set; }     // SMTP
şifresi
    public string FromEmail { get; set; }        //
Gönderen e-posta adresi
    public string FromName { get; set; }         //
Gönderen adı
}
```

Açıklama: Bu sınıflar, appsettings.json'daki yapılandırma değerlerini C# nesnelerine dönüştürmek için kullanılır. Her özellik, ilgili yapılandırma değerine karşılık gelir.

1.3 Program.cs Konfigürasyonu

1.3.1 Dependency Injection

```
// Servis kayıtları
builder.Services.AddScoped<UserService>();       // Her
HTTP isteği için yeni bir UserService örneği
builder.Services.AddScoped<JwtService>();        // JWT
işlemleri için servis
builder.Services.AddScoped<LogService>();        //
Loglama işlemleri için servis
builder.Services.AddScoped<DatabaseSeeder>();    //
Veritabanı başlangıç verileri için servis

// Identity servisleri
builder.Services.AddIdentity<ApplicationUser,
IdentityRole>(options =>
{
    options.Password.RequireDigit = true;
// Şifrede en az bir rakam zorunlu
    options.Password.RequireLowercase = true;
// Şifrede en az bir küçük harf zorunlu
    options.Password.RequireUppercase = true;
// Şifrede en az bir büyük harf zorunlu
    options.Password.RequireNonAlphanumeric = true;
// Şifrede en az bir özel karakter zorunlu
    options.Password.RequiredLength = 8;
// Minimum şifre uzunluğu
})
```

```
.AddEntityFrameworkStores<AppDbContext>()
// Identity verilerini AppDbContext'te sakla
.AddDefaultTokenProviders();                                    //
Varsayılan token sağlayıcılarını ekle
```

Açıklama: Bu kod, uygulamanın bağımlılık enjeksiyon konteynerini yapılandırır. Her servis, yaşam döngüsüne göre kaydedilir ve Identity framework'ü için gerekli servisler eklenir.

1.3.2 Middleware Konfigürasyonu

```
// Middleware sırası önemlidir
app.UseRouting();           // Rota eşleştirme
middleware'i
app.UseAuthentication();    // Kimlik doğrulama
middleware'i
app.UseAuthorization();     // Yetkilendirme middleware'i
app.MapControllers();       // Controller'ları eşleştir
```

Açıklama: Middleware'ler, HTTP isteklerinin işlenme sırasını belirler. Sıralama önemlidir çünkü her middleware bir sonraki middleware'e geçmeden önce işini tamamlamalıdır.

Bölüm 2: Veri Modelleri ve Entity Framework

2.1 Veri Modelleri ve İlişkiler

2.1.1 ApplicationUser Modeli

```
public class ApplicationUser : IdentityUser
{
    [Required]
    [StringLength(50)]
    public string FirstName { get; set; }        //
Kullanıcının adı

    [Required]
    [StringLength(50)]
    public string LastName { get; set; }         //
Kullanıcının soyadı

    // Navigation properties
    public virtual ICollection<Log> Logs { get; set; }  //
```

```
    Kullanıcının log kayıtları
    }
```

Açıklama: `ApplicationUser` sınıfı, ASP.NET Core Identity'nin temel `IdentityUser` sınıfını genişleterek ek özellikler ekler. `[Required]` ve `[StringLength]` attribute'ları veri doğrulama kurallarını belirtir.

2.1.2 Log Modeli

```
public class Log
{
    public int Id { get; set; }                 // Log kaydının benzersiz kimliği
    public string Level { get; set; }           // Log seviyesi (Error, Warning, Info vb.)
    public string Message { get; set; }         // Log mesajı
    public string Exception { get; set; }       // Hata detayları
    public string Source { get; set; }          // Log kaynağı
    public string Action { get; set; }          // Gerçekleştirilen işlem
    public string UserId { get; set; }          // İşlemi gerçekleştiren kullanıcı
    public string IpAddress { get; set; }       // İşlemin yapıldığı IP adresi
    public DateTime CreatedAt { get; set; }     // Log oluşturulma zamanı

    // Navigation property
    public virtual ApplicationUser User { get; set; }  // İlgili kullanıcı
}
```

Açıklama: `Log` sınıfı, sistemdeki tüm önemli olayları kaydetmek için kullanılır. Her log kaydı, kim tarafından, ne zaman ve hangi IP adresinden yapıldığı bilgisini içerir.

2.2 DbContext Yapılandırması

2.2.1 AppDbContext

```
public class AppDbContext :
IdentityDbContext<ApplicationUser>
{
    public AppDbContext(DbContextOptions<AppDbContext>
options) : base(options)
    {
    }

    public DbSet<Log> Logs { get; set; }  // Log tablosu
için DbSet

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Role seed data
        modelBuilder.Entity<IdentityRole>().HasData(
            new IdentityRole { Id = "1", Name = "Admin",
NormalizedName = "ADMIN" },
            new IdentityRole { Id = "2", Name = "User",
NormalizedName = "USER" }
        );

        // Log entity configuration
        modelBuilder.Entity<Log>(entity =>
        {
            entity.Property(e => e.CreatedAt)
                .HasDefaultValueSql("GETUTCDATE()");  //
Varsayılan olarak UTC zamanı kullan

            entity.HasOne(e => e.User)
                .WithMany(u => u.Logs)
                .HasForeignKey(e => e.UserId)
                .OnDelete(DeleteBehavior.Restrict);  //
Kullanıcı silindiğinde loglar silinmez
        });
    }
}
```

Açıklama: `AppDbContext` sınıfı, veritabanı işlemlerini yönetir. `OnModelCreating` metodu, veritabanı şemasını ve ilişkileri yapılandırır. Ayrıca, başlangıç rolleri ve varsayılan değerler tanımlanır.

2.2.2 Migration Stratejisi

```
# Yeni migration oluşturma
dotnet ef migrations add InitialCreate
```

```
# Migration'ları uygulama
dotnet ef database update
```

Açıklama: Bu komutlar, veritabanı şemasını güncellemek için kullanılır. `InitialCreate` migration'ı, veritabanını ilk kez oluştururken kullanılır.

Bölüm 3: Authentication ve Authorization

3.1 JWT Authentication

3.1.1 JWT Konfigürasyonu

```
// JWT ayarlarını alma
var jwtSettings =
builder.Configuration.GetSection("JwtSettings");
var key =
Encoding.UTF8.GetBytes(jwtSettings["SecretKey"]);

// Authentication servislerini ekleme
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new
TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,         // Token
imzasını doğrula
        IssuerSigningKey = new SymmetricSecurityKey(key),
// İmza doğrulama anahtarı
        ValidateIssuer = true,                   // Token
yayıncısını doğrula
        ValidIssuer = jwtSettings["Issuer"],     //
Geçerli yayıncı
        ValidateAudience = true,                 // Token
hedef kitlesini doğrula
        ValidAudience = jwtSettings["Audience"], //
Geçerli hedef kitle
        ValidateLifetime = true,                 // Token
süresini doğrula
        ClockSkew = TimeSpan.Zero                // Zaman
farkı toleransı
```

```
    };
});
```

Açıklama: Bu kod, JWT tabanlı kimlik doğrulama sistemini yapılandırır. Her token'ın geçerliliği, imzası, yayıncısı ve hedef kitlesi kontrol edilir.

3.1.2 JWT Token Üretimi

```
public string GenerateAccessToken(ApplicationUser user)
{
    var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id),
// Kullanıcı ID'si
        new Claim(ClaimTypes.Name, user.UserName),
// Kullanıcı adı
        new Claim(ClaimTypes.Email, user.Email),
// E-posta adresi
        new Claim(ClaimTypes.Role,
user.Roles.FirstOrDefault()?.Name ?? "User")  // Kullanıcı
rolü
    };

    var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_secretKey));
    var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: _issuer,
// Token yayıncısı
        audience: _audience,
// Hedef kitle
        claims: claims,
// Kullanıcı bilgileri
        expires:
DateTime.UtcNow.AddMinutes(_accessTokenExpirationMinutes),
// Son kullanma tarihi
        signingCredentials: creds
// İmza bilgileri
    );

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

Açıklama: Bu metod, kullanıcı bilgilerini içeren bir JWT token oluşturur. Token, kullanıcının kimliğini ve yetkilerini içerir ve belirli bir süre için geçerlidir.

3.2 Authorization

3.2.1 Policy Tanımları

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("AdminOnly", policy =>
        policy.RequireRole("Admin"));           // Sadece
Admin rolüne sahip kullanıcılar

    options.AddPolicy("UserOnly", policy =>
        policy.RequireRole("User"));            // Sadece
User rolüne sahip kullanıcılar

    options.AddPolicy("AdminOrUser", policy =>
        policy.RequireRole("Admin", "User"));   // Admin
veya User rolüne sahip kullanıcılar
});
```

Açıklama: Bu kod, farklı yetkilendirme politikalarını tanımlar. Her politika, belirli rollerdeki kullanıcıların erişimini kontrol eder.

3.2.2 Controller Seviyesinde Authorization

```
[ApiController]
[Route("api/[controller]")]
[Authorize(Policy = "AdminOnly")]  // Sadece Admin rolüne
sahip kullanıcılar erişebilir
public class UserController : ControllerBase
{
    // ...
}
```

Açıklama: `[Authorize]` attribute'u, bir controller veya action'a erişimi kısıtlar. Bu örnekte, sadece Admin rolüne sahip kullanıcılar `UserController`'a erişebilir.

Bölüm 4: API Endpoints ve Controller'lar

4.1 UserController

4.1.1 Get Endpoints

```
[HttpGet("list")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
public async Task<IActionResult> GetAllUsers()
{
    var users = await _userService.GetAllUsersAsync();
    return Ok(users);
}

[HttpGet("{id:guid}")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetUserById(Guid id)
{
    var user = await _userService.GetUserByIdAsync(id);
    if (user == null) return NotFound();
    return Ok(user);
}
```

Açıklama: Bu endpoint'ler, kullanıcı listesini ve belirli bir kullanıcının detaylarını getirir. `[ProducesResponseType]` attribute'ları, olası HTTP yanıt kodlarını belgeler.

4.1.2 Post Endpoints

```
[AllowAnonymous]
[HttpPost("register")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> CreateUser(RegisterDto
registerDto)
{
    var user = await _userService.CreateUserAsync(
        registerDto.Username,
        registerDto.Password,
        registerDto.Email,
        registerDto.FirstName,
        registerDto.LastName,
        registerDto.PhoneNumber
    );
    return CreatedAtAction(nameof(GetUserById), new { id =
user.Id }, user);
}
```

```
[HttpPost("create-admin")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
public async Task<IActionResult>
CreateAdminUser(RegisterDto registerDto)
{
    var user = await _userService.CreateUserAsync(
        registerDto.Username,
        registerDto.Password,
        registerDto.Email,
        registerDto.FirstName,
        registerDto.LastName,
        registerDto.PhoneNumber,
        roleId: 1
    );
    return CreatedAtAction(nameof(GetUserById), new { id =
user.Id }, user);
}
```

Açıklama: Bu endpoint'ler, yeni kullanıcı ve admin kullanıcı oluşturur. `[AllowAnonymous]` attribute'u, kayıt işleminin kimlik doğrulaması gerektirmediğini belirtir.

4.2 DTOs (Data Transfer Objects)

4.2.1 RegisterDto

```
public class RegisterDto
{
    [Required]
    [StringLength(50)]
    public string Username { get; set; }         //
Kullanıcı adı

    [Required]
    [StringLength(100)]
    public string Password { get; set; }         // Şifre

    [Required]
    [EmailAddress]
    [StringLength(100)]
    public string Email { get; set; }            // E-
posta adresi

    [StringLength(50)]
    public string FirstName { get; set; }        // Ad
```

```
    [StringLength(50)]
    public string LastName { get; set; }         // Soyad

    [StringLength(20)]
    public string PhoneNumber { get; set; }      //
Telefon numarası
}
```

Açıklama: `RegisterDto` sınıfı, kullanıcı kaydı için gerekli verileri taşır. Her özellik, veri doğrulama kurallarıyla birlikte tanımlanır.

Bölüm 5: Servis Katmanı ve İş Mantığı

5.1 UserService

5.1.1 Kullanıcı Yönetimi

```
public class UserService
{
    private readonly AppDbContext _context;
    private readonly UserManager<ApplicationUser>
_userManager;
    private const int MAX_FAILED_ATTEMPTS = 5;
// Maksimum başarısız giriş denemesi
    private const int LOCKOUT_DURATION_MINUTES = 15;
// Hesap kilitleme süresi

    public UserService(
        AppDbContext context,
        UserManager<ApplicationUser> userManager)
    {
        _context = context;
        _userManager = userManager;
    }

    public async Task<List<ApplicationUser>>
GetAllUsersAsync()
    {
        return await _context.Users
            .Include(u => u.Logs)
// Kullanıcının log kayıtlarını da getir
            .ToListAsync();
    }

    public async Task<ApplicationUser?>
GetUserByIdAsync(string userId)
    {
```

```
        return await _context.Users
            .Include(u => u.Logs)
// Kullanıcının log kayıtlarını da getir
            .FirstOrDefaultAsync(u => u.Id == userId);
    }
}
```

Açıklama: `UserService` sınıfı, kullanıcı yönetimi işlemlerini gerçekleştirir. Veritabanı işlemleri ve kullanıcı doğrulama işlemleri burada yapılır.

5.1.2 Kullanıcı Doğrulama

```
public async Task<ApplicationUser?>
AuthenticateUser(string username, string password)
{
    var user = await
_userManager.FindByNameAsync(username);
    if (user == null) return null;

    var result = await
_userManager.CheckPasswordAsync(user, password);
    if (!result)
    {
        user.AccessFailedCount++;
// Başarısız giriş sayısını artır
        if (user.AccessFailedCount >= MAX_FAILED_ATTEMPTS)
        {
            user.LockoutEnd =
DateTime.UtcNow.AddMinutes(LOCKOUT_DURATION_MINUTES);  //
Hesabı kilitle
        }
        await _context.SaveChangesAsync();
        return null;
    }

    user.AccessFailedCount = 0;
// Başarılı girişte sayaçı sıfırla
    user.LastLoginAt = DateTime.UtcNow;
// Son giriş zamanını güncelle
    await _context.SaveChangesAsync();
    return user;
}
```

Açıklama: Bu metod, kullanıcı girişini doğrular ve güvenlik önlemlerini uygular. Başarısız giriş denemelerini takip eder ve hesabı geçici olarak kilitleyebilir.

5.2 DatabaseSeeder

5.2.1 Veritabanı Başlatma

```
public class DatabaseSeeder
{
    private readonly AppDbContext _context;
    private readonly UserManager<ApplicationUser>
_userManager;
    private readonly RoleManager<IdentityRole>
_roleManager;
    private readonly IConfiguration _configuration;

    public async Task SeedAsync()
    {
        await _context.Database.MigrateAsync();
// Veritabanını güncelle

        if (!await _roleManager.RoleExistsAsync("Admin"))
        {
            await _roleManager.CreateAsync(new
IdentityRole("Admin"));  // Admin rolünü oluştur
        }

        var adminSettings =
_configuration.GetSection("AdminUser").Get<AdminUserSettin
gs>();
        if (adminSettings == null)
        {
            throw new InvalidOperationException("Admin
user settings not found");
        }

        var adminUser = await
_userManager.FindByEmailAsync(adminSettings.Email);
        if (adminUser == null)
        {
            adminUser = new ApplicationUser
            {
                UserName = adminSettings.Username,
                Email = adminSettings.Email,
                FirstName = adminSettings.FirstName,
                LastName = adminSettings.LastName,
                PhoneNumber = adminSettings.PhoneNumber,
                EmailConfirmed = true
            };

            var result = await
_userManager.CreateAsync(adminUser,
adminSettings.Password);
            if (!result.Succeeded)
```

```
            {
                throw new InvalidOperationException(
                    $"Failed to create admin user:
{string.Join(", ", result.Errors.Select(e =>
e.Description))}");
            }

            await _userManager.AddToRoleAsync(adminUser,
"Admin");  // Admin rolünü ata
        }
    }
}
```

Açıklama: `DatabaseSeeder` sınıfı, uygulama ilk kez çalıştığında veritabanını hazırlar. Admin rolünü ve varsayılan admin kullanıcısını oluşturur.

Bölüm 6: Güvenlik ve Hata Yönetimi

6.1 Global Error Handling

6.1.1 Exception Middleware

```
app.Use(async (context, next) =>
{
    try
    {
        await next();
    }
    catch (Exception ex)
    {
        var logService =
context.RequestServices.GetRequiredService<LogService>();
        var userId =
context.User?.FindFirst(ClaimTypes.NameIdentifier)?.Value;
        var ipAddress =
context.Connection?.RemoteIpAddress?.ToString();

        await logService.LogAsync(
            level: "Error",
            message: ex.Message,
            exception: ex.ToString(),
            source: ex.Source,
            action: context.Request.Path,
            userId: userId,
            ipAddress: ipAddress
        );

        context.Response.StatusCode = 500;
```

```
        await context.Response.WriteAsJsonAsync(new {
error = "An error occurred while processing your request."
});
    }
});
```

Açıklama: Bu middleware, uygulamada oluşan tüm hataları yakalar, loglar ve kullanıcıya uygun bir hata mesajı döner.

6.1.2 Custom Exception Handler

```
public class ApiException : Exception
{
    public int StatusCode { get; set; }          // HTTP
durum kodu
    public string ErrorCode { get; set; }        // Hata
kodu

    public ApiException(string message, int statusCode =
500, string errorCode = "INTERNAL_ERROR")
        : base(message)
    {
        StatusCode = statusCode;
        ErrorCode = errorCode;
    }
}
```

Açıklama: `ApiException` sınıfı, API'ye özel hata türlerini temsil eder. Her hata, bir HTTP durum kodu ve hata kodu içerir.

6.2 Güvenlik Önlemleri

6.2.1 Password Hashing

```
public static class PasswordHasher
{
    public static string HashPassword(string password)
    {
        return BCrypt.Net.BCrypt.HashPassword(password);
// Şifreyi hash'le
    }

    public static bool VerifyPassword(string password,
```

```
string hash)
    {
        return BCrypt.Net.BCrypt.Verify(password, hash);
// Şifreyi doğrula
    }
}
```

Açıklama: Bu sınıf, şifreleri güvenli bir şekilde hash'lemek ve doğrulamak için BCrypt algoritmasını kullanır.

6.2.2 Rate Limiting

```
builder.Services.AddRateLimiter(options =>
{
    options.GlobalLimiter =
PartitionedRateLimiter.Create<HttpContext, string>(context
=>
        RateLimitPartition.GetFixedWindowLimiter(
            partitionKey: context.User.Identity?.Name ??
context.Request.Headers.Host.ToString(),
            factory: partition => new
FixedWindowRateLimiterOptions
            {
                AutoReplenishment = true,         // Limit
otomatik olarak yenilensin
                PermitLimit = 100,                // 1
dakikada maksimum 100 istek
                Window = TimeSpan.FromMinutes(1)  // 1
dakikalık pencere
            }));
});
```

Açıklama: Bu kod, API'ye yapılan istekleri sınırlandırır. Her kullanıcı veya IP adresi için belirli bir sürede maksimum istek sayısı belirlenir.

Bölüm 7: Swagger/OpenAPI Dokümantasyonu

7.1 Swagger Konfigürasyonu

7.1.1 Swagger Servis Kaydı

```
builder.Services.AddSwaggerGen(c =>
{
```

```
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "FreeBirds API",
        Version = "v1",
        Description = "FreeBirds API Documentation",
        Contact = new OpenApiContact
        {
            Name = "FreeBirds Team",
            Email = "support@freebirds.com"
        }
    });

    c.AddSecurityDefinition("Bearer", new
OpenApiSecurityScheme
    {
        Description = "JWT Authorization header using the
Bearer scheme",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer"
    });

    c.AddSecurityRequirement(new
OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            new string[] {}
        }
    });
});
```

Açıklama: Bu kod, Swagger UI'ı yapılandırır ve JWT kimlik doğrulaması için gerekli ayarları ekler.

7.1.2 Swagger UI Konfigürasyonu

```
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
```

```
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json",
"FreeBirds API V1");
        c.RoutePrefix = "swagger";
        c.DocumentTitle = "FreeBirds API Documentation";
        c.DefaultModelsExpandDepth(-1);  // Şema bölümünü
gizle
    });
}
```

Açıklama: Bu kod, Swagger UI'ı sadece geliştirme ortamında etkinleştirir ve görünümünü özelleştirir.

7.2 API Dokümantasyonu

7.2.1 XML Comments

```
/// <summary>
/// Creates a new user in the system
/// </summary>
/// <param name="registerDto">User registration
information</param>
/// <returns>Created user information</returns>
/// <response code="201">Returns the newly created
user</response>
/// <response code="400">If the registration data is
invalid</response>
[HttpPost("register")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> CreateUser(RegisterDto
registerDto)
```

Açıklama: XML yorumları, API endpoint'lerinin dokümantasyonunu oluşturmak için kullanılır. Swagger UI'da görüntülenir.

Bölüm 8: Deployment ve Production Hazırlığı

8.1 Production Konfigürasyonu

8.1.1 Environment-specific Ayarlar

```
// appsettings.Production.json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=production-
db;Database=FreeBirds;User
Id=sa;Password=***;TrustServerCertificate=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

Açıklama: Bu dosya, production ortamı için özel ayarları içerir. Veritabanı bağlantı bilgileri ve loglama seviyeleri burada tanımlanır.

8.1.2 Logging Konfigürasyonu

```
builder.Logging.AddFile("Logs/freebirds-{Date}.txt",
LogLevel.Warning);
```

Açıklama: Bu kod, log mesajlarını dosyaya yazmak için gerekli yapılandırmayı ekler. Her gün için ayrı bir log dosyası oluşturulur.

8.2 Deployment Stratejisi

8.2.1 Dockerfile

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["FreeBirds.csproj", "./"]
RUN dotnet restore "FreeBirds.csproj"
COPY . .
WORKDIR "/src"
RUN dotnet build "FreeBirds.csproj" -c Release -o
/app/build
```

```
FROM build AS publish
RUN dotnet publish "FreeBirds.csproj" -c Release -o
/app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "FreeBirds.dll"]
```

Açıklama: Bu Dockerfile, uygulamayı bir Docker konteynerinde çalıştırmak için gerekli adımları tanımlar. Çok aşamalı bir build süreci kullanır.

8.2.2 CI/CD Pipeline

```
# .github/workflows/deploy.yml
name: Deploy FreeBirds API

on:
  push:
    branches: [ main ]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    - name: Setup .NET
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: '8.0.x'

    - name: Build
      run: dotnet build --configuration Release

    - name: Test
      run: dotnet test

    - name: Publish
      run: dotnet publish --configuration Release --output
./publish

    - name: Deploy to Azure
      uses: azure/webapps-deploy@v2
      with:
        app-name: 'freebirds-api'
        publish-profile: ${{ secrets.AZURE_PUBLISH_PROFILE
```

```
}}
        package: ./publish
```

Açıklama: Bu GitHub Actions workflow'u, uygulamayı otomatik olarak derler, test eder ve Azure'a dağıtır. Her `main` branch'ine push yapıldığında tetiklenir.

Author

Name: Hüseyin Akçiçek
Contact: huseyin.akcicek@outlook.com