**Field Expense Manager API Dokümantasyon**

**Projeye Genel Bakış**

Bu proje, şirketlerin saha personelinin masraf takibini ve yönetimini modern web teknolojileriyle otomatize etmek amacıyla geliştirilmiş bir Restful API'dir. API, personelin masraflarını kolayca girmesini, yöneticilerin bu masrafları etkin bir şekilde onaylamasını veya reddetmesini ve tüm sürecin raporlanmasını sağlar. Temel hedef, manuel süreçleri azaltmak, ödeme süreçlerini hızlandırmak ve masraf yönetimi verimliliğini artırmaktır.

**Proje İsterleri ve Karşılanma Durumu**

Aşağıda projenin ana isterleri ve bu isterlerin API tarafından nasıl karşılandığı özetlenmiştir:

1. **Rol Bazlı Erişim (Admin/Personnel):**
   * Sistemde iki farklı rol kullanıldı ve yetkiler buna göre ayrıldı.Roller Admin ve Personnel olarak enum şekilinde saklandı.
   * JWT (JSON Web Tokens) tabanlı kimlik doğrulama ve rol bazlı yetkilendirme sistemi kurulmuştur. API endpoint'leri [Authorize] ve [Authorize(Roles = "...")] attribute'ları ile korunarak, sadece yetkili rollerin ilgili işlemleri yapması sağlanmıştır (AuthController, ExpensesController, ExpenseCategory,UsersController, ReportsController).
2. **Personel İşlemleri:**
   * Personel kendi masrafını girebilmeli, listeleyebilmeli, takip edebilmeli, red sebebini görebilmeli işlemleri karşılanmıştır.
   * **Karşılanma:** /api/expenses (POST),
   * /api/expenses/my (GET),
   * /api/expenses/{id} (GET)

Endpoint'leri ve ilgili CQRS Handler'ları (ExpenseCommandHandler, ExpenseQueryHandler) ile bu işlemler gerçekleştirilmiştir. Personel yetkilendirmesi uygulanmıştır. Personelin girdiği masrafa göre göre ilgili masrafın onaylanması veya reddedilmesi sağlanabilir.ExpenseDto içindeki rejectionReason red sebebini içerir.Reddedilen masrafın ödemesi de yapılmaz , ExpenseDto içindeki payments bölümü null gelir.Eğer masraf onaylanmışsa bu bölüm masrafa özel TransactionId,masrafın yapıldığı tarih PaymentDate ve IsSuccessful kısmı başarı olarak dolu gelir.

1. **Admin İşlemleri:**
   * Admin tüm masrafları görebilmeli, onay/red yapabilmeli (sebep ile), kullanıcıları ve kategorileri yönetebilmeli, raporları alabilmeli gibi durumlar sağlanmıştır.
   * **Karşılanma:**
   * /api/expenses (GET),
   * /api/expenses/{id}/approve (PUT),
   * /api/expenses/{id}/reject (PUT),
   * /api/expenses/{id} (DELETE - Soft Delete),
   * /api/ExpenseCategory (CRUD),
   * /api/Users (CRUD),
   * /api/Reports endpoint'leri ve ilgili Handler'lar ile tüm bu yetkiler sağlanmıştır. Admin rolü yetkilendirmesi uygulanmıştır.
2. **Masraf Detayları ve Ekler:**
   * Masraflar kategorili olmalı, ek (fiş/fatura) yüklenebilmeli gibi özellikler sağlanmıştır.
   * Expense entity'si ExpenseCategory ile ilişkilendirilmiştir. Masraf oluşturma (CreateExpenseCommand) IFormFile kabul eder ve ExpenseCommandHandler dosyayı sunucuya kaydeder (wwwroot/uploads), bilgileri ExpenseAttachment olarak veritabanına yazar. Okuma işlemlerinde ek bilgileri DTO ile döndürülür.
3. **Ödeme Simülasyonu:**
   * Onaylanan masraflar için hayali/simüle bir ödeme sistemi yapıldı.
   * IPaymentService arayüzü ve FakeBankPaymentService implementasyonu oluşturulmuştur. ApproveExpenseCommandHandler, masrafı onayladıktan sonra bu servisi çağırır ve başarılı/başarısız ödeme durumu ExpensePayments tablosuna loglanır.
4. **Raporlama (Dapper):**
   * Belirtilen 4 ana rapor (Personel Geçmişi, Ödeme Yoğunluğu, Personel Harcaması, Onay/Red Durumu) Dapper kullanılarak günlük/haftalık/aylık periyotlarda alınabilecek şekilde oluşturuldu.
   * IReportingRepository ve ReportingRepository oluşturulmuştur. İlgili metotlar Dapper kullanarak SQL sorguları çalıştırır ve sonuçları özel Rapor DTO'larına dönüştürür. ReportsController ve ilgili CQRS Query/Handler'ları ile bu raporlar sunulur.
5. **Teknik İsterler:**
   * **JWT:** AuthService ve ASP.NET Core Authentication/Authorization middleware'leri ile JWT üretimi ve doğrulaması sağlanmıştır.
   * **EF Core - Repository - Unit of Work:** Veritabanı işlemleri için bu desenler kullanılmıştır. GenericRepository temel işlemleri, özel Repository'ler (örn: UserRepository) entity'ye özgü işlemleri sağlanmıştır. UnitOfWork ile transaction yönetimini ve merkezi repository erişimi yönetilmiştir.
   * **Swagger/Postman:** Swagger UI entegrasyonu yapılmıştır. Postman koleksiyonu oluşturulmuştur.
   * **Code First:** Proje Code First yaklaşımıyla geliştirilmiş, veritabanı şeması Migration'lar ile yönetilmiştir.
   * **Validasyon:** FluentValidation ile DTO seviyesinde temel doğrulamalar, Handler ile ise iş kuralı doğrulamaları yapılmıştır.
   * **REST Standartları:** Uygun HTTP metotları (GET, POST, PUT, DELETE), durum kodları (200, 201, 204, 400, 401, 403, 404, 500) ve route yapıları kullanılmıştır.
   * **Kod Kalitesi Kriterleri:** Anlamlı isimlendirme, SOLID prensiplerine (özellikle Tek Sorumluluk), Dependency Injection, düşük karmaşıklık gibi kriterlere dikkat edilmiştir.

**Kullanılan Teknolojiler (Tech Stack)**

* **Backend Framework:** .NET 8 (ASP.NET Core Web API)
* **Veritabanı:** SQL Server (EF Core Code First ile yönetilir)
* **ORM:** Entity Framework Core 8
* **Micro ORM:** Dapper (Raporlama için)
* **Mimari Desenler:** Katmanlı Mimari (Domain, Application, Infrastructure, API), CQRS (Command Query Responsibility Segregation) prensibi (MediatR kütüphanesi ile), Repository ve Unit of Work Desenleri.
* **Yetkilendirme:** JWT (JSON Web Tokens)
* **Validasyon:** FluentValidation
* **Object Mapping:** AutoMapper
* **API Dokümantasyon & Test:** Swagger (Swashbuckle.AspNetCore) ve Postman
* **Loglama:** Microsoft.Extensions.Logging (ASP.NET Core yerleşik)

**Mimari Genel Bakış**

Proje, modern yazılım geliştirme prensiplerine uygun olarak tasarlanmıştır:

1. **Katmanlı Mimari:** Sorumluluklar net bir şekilde ayrılmıştır:
   * **Domain:** İş kurallarının ve temel varlıkların (Entity, Enum) bulunduğu çekirdek katman oluşturur.
   * **Application:** İş akışlarının yönetildiği, CQRS komut/sorgularının, Handler'ların, DTO'ların, arayüzlerin, validasyonların ve AutoMapper profillerinin bulunduğu katmandır. İş mantığı buradadır.
   * **Infrastructure:** Dış dünya ile etkileşimin (veritabanı erişimi, dosya sistemi, dış servisler) yönetildiği katman. EF Core DbContext, Repository implementasyonları, Dapper kullanımı, UnitOfWork implementasyonu, Seed Data ve Servis implementasyonları (Ödeme) burada yer alır.
   * **API (Presentation):** Uygulamanın dış dünyaya açılan yüzü. HTTP isteklerini alır, yetkilendirmeyi kontrol eder, MediatR aracılığıyla uygun Handler'a yönlendirir ve sonucu HTTP yanıtına çevirir. Controller'lar ve Middleware bu katmandadır.
2. **CQRS (MediatR ile):** Okuma (Query) ve Yazma/Güncelleme (Command) işlemleri birbirinden ayrılmıştır. Her Command ve Query'nin kendi özel Handler'ı bulunur. Bu, kodun daha odaklı, test edilebilir ve yönetilebilir olmasını sağlar. MediatR kütüphanesi, isteklerin doğru Handler'a yönlendirilmesini kolaylaştırır.
3. **Repository ve Unit of Work:** Veri erişim mantığı Repository deseni ile soyutlanmıştır (IGenericRepository ve özel arayüzler). Unit of Work deseni, tüm veritabanı değişikliklerinin tek bir transaction içinde tutarlı bir şekilde kaydedilmesini garanti eder (CompleteAsync) ve Repository'lere merkezi erişim sağlar.
4. **Dependency Injection:** Katmanlar ve sınıflar arasındaki bağımlılıklar, .NET Core'un yerleşik DI konteyneri kullanılarak yönetilir. Bu, kodun esnekliğini ve test edilebilirliğini artırır.

**Kurulum Adımları**

1. **Ön Gereksinimler:**
   * .NET 8 SDK
   * SQL Server & SSMS
   * Postman
2. **Projeyi Klonlama:** git clone <repository\_url>
3. **Veritabanı Kurulumu:**
   * appsettings.Development.json içindeki ConnectionStrings:DefaultConnection'ı kendi SQL Server bilgilerinize göre güncelleyin.
   * API projesi dizininde;

* dotnet ef database update

--project FieldExpenseManager.Infrastructure

--startup-project FieldExpenseManager.Api komutunu çalıştırın.

1. **Bağımlılıkları Yükleme:** Proje kök dizininde dotnet restore komutunu çalıştırın.

**Yapılandırma (Configuration)**

* **Veritabanı:** appsettings.Development.json -> ConnectionStrings:DefaultConnection kısımını kendi veritabanı bilgilerinize göre düzenleyin.
* **JWT:** appsettings.Development.json -> JwtSettings :Kendi ortamınıza göre düzenleyin.

**Uygulamayı Çalıştırma**

* API projesi dizininde dotnet run komutu veya Visual Studio'dan F5.
* Çalışan adresler konsolda belirtilecektir (örn: https://localhost:7234).

**API Kullanımı ve Endpoint Açıklamaları**

API endpoint'lerinin detaylı açıklamaları ve test arayüzü için uygulama çalışırken /swagger adresini ziyaret ediniz veya hazırladığım postman collectionu üzerinden test edebilirsiniz

**Temel Akış:**

1. **Login Ol:** /api/auth/login endpoint'ine POST isteği ile SeedData Klasörünün içindeki UserDataSeeder dosyasından default kullanıcıların, kullanıcı adı ve şifresini gönderin.
2. **Token Al:** Başarılı girişte dönen JWT token'ını kopyalayın.
3. **Yetkilendirme:** Swagger UI'daki "Authorize" butonuna tıklayıp Bearer <token> formatında token'ı yapıştırın ve yetkilendirin.
4. **Diğer Endpoint'leri Kullan:** Artık yetki gerektiren endpoint'lere istek gönderebilirsiniz.

**Endpoint Grupları:**

**Genel Endpoint'ler**

* **Kullanıcı Girişi-LOGİN**
  + **Amaç:** Sisteme giriş yaparak JWT token almak.
  + **Metot/Route:** POST /api/auth/login
  + **Yetki:** Yok (Herkese Açık)
  + **Parametreler:**
    - **Request Body:** LoginDto (Email, Password içerir)

**Personel Yetkileri (**[Authorize(Roles = "Personnel")] **veya** [Authorize]**)**

* **Yeni Masraf Oluşturma-EXPENSE**
  + **Amaç:** Sisteme yeni bir masraf kaydı ve ek dosyasını girmek.
  + **Metot/Route:** POST /api/Expenses
  + **Yetki:** Yanlızca Personel
  + **Parametreler:**
    - **Request Body (form-data):**
      * ExpenseCategoryId (int, Zorunlu): Kategori ID'si.
      * Description (string, Zorunlu): Açıklama.
      * Amount (decimal, Zorunlu): Tutar.
      * ExpenseDate (DateTime, Zorunlu): Masraf tarihi.
      * Location (string, Opsiyonel): Konum.
      * attachmentFile (IFormFile, Opsiyonel): Yüklenecek dosya.
* **Kendi Masraflarını Listeleme**
  + **Amaç:** Giriş yapmış personelin sadece kendisine ait masrafları listelemek.
  + **Metot/Route:** GET /api/Expenses/my
  + **Yetki:** Personel (Admin de erişebilir)
  + **Parametreler:** Yok
* **Kendi Masraf Detayını Görme**
  + **Amaç:** Giriş yapmış personelin kendisine ait belirli bir masrafın detaylarını görmek.
  + **Metot/Route:** GET /api/Expense/ExpenseById/{id}
  + **Yetki:** Personel (Sadece kendi masrafı için görebilir) veya Admin
  + **Parametreler:**
    - id (int, Route): Görüntülenecek masrafın ID'si.
* **Kendi Masraf Geçmişi Raporu**
  + **Amaç:** Giriş yapmış personelin masraf geçmişini (tarih filtresiyle) Dapper kullanarak almak.
  + **Metot/Route:** GET /api/Reports/MyExpenseHistory
  + **Yetki:** Yanlızca Personel
  + **Parametreler:**
    - startDate : Başlangıç tarihi.
    - endDate : Bitiş tarihi.

**Admin Yetkileri (**[Authorize(Roles = "Admin")]**)**

* **Kategori Yönetimi (CRUD)-EXPENSECATEGORY**
  + **Amaç:** Masraf kategorilerini listelemek, eklemek, güncellemek, silmek (soft delete).
  + **Metot/Route:**
    - GET /api/ExpenseCategory: Tüm aktif kategorileri listeler (Bu herkese açıktır ancak en azından login gerektirir- Controller'da [Authorize] var).
    - GET /api/ExpenseCategory/{id}: Belirli bir kategoriyi getirir (Bu en azından login gerektirir).
    - POST /api/ExpenseCategory: Yeni kategori ekler. Body: CreateCategoryDto.(Bu İşlem yanlızca Admin tarafından yapılabilir.)
    - PUT /api/ExpenseCategory/{id}: Mevcut kategoriyi günceller. Body: UpdateCategoryDto.(Bu İşlem yanlızca Admin tarafından yapılabilir.)
    - DELETE /api/ExpenseCategory/{id}: Kategoriyi pasif hale getirir (Soft Delete).(Bu İşlem yanlızca Admin tarafından yapılabilir.)
  + **Yetki:** POST, PUT, DELETE için Admin. GET,GETBYID için en azından login.
* **Kullanıcı Yönetimi (CRUD)-USER**
  + **Amaç:** Kullanıcıları (özellikle Personelleri) listelemek, eklemek, güncellemek, silmek (soft delete).
  + **Metot/Route:**
    - GET /api/Users: Tüm kullanıcıları listeler.
    - GET /api/Users/{id}: Belirli bir kullanıcıyı getirir.
    - POST /api/Users: Yeni kullanıcı ekler. Body: CreateUserDto.
    - PUT /api/Users/{id}: Mevcut kullanıcıyı günceller. Body: UpdateUserDto.
    - DELETE /api/Users/{id}: Kullanıcıyı pasif hale getirir (Soft Delete).
  + **Yetki:** Tüm metotlar için Admin yetkisi gereklidir.
* **Tüm Masrafları Listeleme**
  + **Amaç:** Sistemdeki tüm masrafları detaylarıyla listelemek.
  + **Metot/Route:** GET /api/Expenses/GetAllPersonnel
  + **Yetki:** Admin,Personal yetkisi yoktur.
  + **Parametreler:** Yok
* **Masraf Detayı Görme (Herhangi Biri)**
  + **Amaç:** ID'si verilen herhangi bir masrafın detaylarını görmek.
  + **Metot/Route:** GET /api/Expenses/ExpenseById/{id}
  + **Yetki:** Admin (Personel sadece kendi masrafını görebilir, bu endpoint'i Admin kullanır-Admin tüm kullanıcıları görebilir).
  + **Parametreler:**
    - id (int, Route): Görüntülenecek masrafın ID'si.
* **Masraf Onaylama**
  + **Amaç:** Beklemede olan bir masrafı onaylamak ve ödeme simülasyonunu tetiklemek.
  + **Metot/Route:** PUT /api/Expenses/{id}/approve
  + **Yetki:** Yalnızca Admin
  + **Parametreler:**
    - id (int, Route): Onaylanacak masrafın ID'si.
* **Masraf Reddetme**
  + **Amaç:** Beklemede olan bir masrafı sebep belirterek reddetmek.
  + **Metot/Route:** PUT /api/Expenses/{id}/reject
  + **Yetki:** Yalnızca Admin
  + **Parametreler:**
    - id (int, Route): Reddedilecek masrafın ID'si.
    - **Request Body:** RejectExpenseDto (Reason içerir).
* **Masraf Silme (Soft Delete)**
  + **Amaç:** Bir masraf kaydını pasif hale getirmek.
  + **Metot/Route:** DELETE /api/Expenses/{id}
  + **Yetki:Yalnızca** Admin
  + **Parametreler:**
    - id (int, Route): Pasif yapılacak masrafın ID'si.
* **Şirket Raporları (Admin)**
  + **Amaç:** Şirket geneli raporları almak (Ödeme Yoğunluğu, Personel Harcaması, Onay/Red Durumu).
  + **Metot/Route:**
    - GET /api/Reports/PaymentDensity
    - GET /api/Reports/PersonnelSpending
    - GET /api/Reports/ApprovalStatus
  + **Yetki:** Admin
  + **Parametreler:** İlgili raporun gerektirdiği startDate, endDate, period, personnelId gibi query parametreleri ([FromQuery]).

**Örnek Kullanıcılar (Seed Data- UserDataSeeder)**

* **Admin:** burcu@gmail.com / AdminSifre123!
* **Personel:** personel@gmail.com / PersonelSifre789.
* **Personel:** [personel2@gmail.com](mailto:personel2@gmail.com) / PersonelSifre7897.

**Validasyon ve Hata Yönetimi**

* **FluentValidation:** Gelen DTO'ların temel format ve değer kuralları Validator sınıfları ile kontrol edilir. Hatalar ValidationException olarak fırlatılır.
* **Handler İş Kuralları:** Veritabanı kontrolü gerektiren (örn: email unique mi?, kayıt silinebilir mi?) validasyonlar Handler içinde yapılır ve uygun standart Exception'lar (InvalidOperationException vb.) fırlatılır.
* **ErrorHandlerMiddleware:** Fırlatılan tüm Exception'ları yakalar. Türüne göre uygun HTTP durum kodunu (400, 404, 500 vb.) ve standart bir JSON hata mesajını istemciye döner. Beklenmedik hataların detayları sadece loglanır.

**Ödeme Simülasyonu ve Soft Delete**

* Onaylanan masraflar için FakeBankPaymentService kullanılarak basit bir ödeme simülasyonu yapılır ve sonucu ExpensePayments tablosuna loglanır.
* Silme işlemleri (Kategori, Kullanıcı, Masraf) fiziksel silme yerine IsActive alanı false yapılarak Soft Delete şeklinde gerçekleştirilir.

**Raporlama (Dapper)**

* Proje isterlerinde belirtilen 4 ana rapor (Personel Masraf Geçmişi, Ödeme Yoğunluğu, Personel Bazlı Harcama, Onay/Red Durumu) için ReportingRepository içinde Dapper kullanılarak SQL sorguları çalıştırılır.
* Sonuçlar özel Rapor DTO'larına map edilir ve ilgili CQRS Query Handler'ları aracılığıyla API'den sunulur.

**DATABASE DİAGRAM**metin, ekran görüntüsü, diyagram, paralel içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.