Kişisel Finans API

Ne yaptım = Sıfırdan başlayıp çalışan, güvenli ve ölçeklenebilir bir RESTful Web API

5 temel aşama

1) Veri modelleme (data modeling) =

ne yaptım? = Uygulamanın temel yapı taşı olan Expense sınıfını (Models klasörüne) oluşturdum

Teknik karşılığı = Bu bizim Entity'imizdir. Veritabanındaki tablonun C# tarafındaki aynasıdır; [Required], [Range] gibi Data Annotations kullanarak veri kurallarını burada belirledim.

2) Veritabanı Bağlantısı (ORM/EF Core)

ne yaptım? = SQL kodları yazmak yerine, C# kodlarını veritabanı diline çeviren Entity Framework Core aracını kurdum. Veritabanı olarak olarak hafif

oldugu için SQLite seçtim.
Teknik karşılığı = Code-first yaklaşımını kullandım. Yani önce kodu yazdım, veritabanını koddan türettim.

3) API Yönetimi (Controller / CRUD)

ne yaptım? = Dış dünyadan gelen istekleri karşılayan Expenses Controller'ı yazdım.
Teknik karşılığı =
GET = Verileri okumak için
POST = Yeni veri eklemek için
PUT / DELETE = Güncelleme ve silme işlemleri için
Ayrıca LINQ kullanarak arama ve filtreleme mantığını (Where, Contains) buraya kurdum.

4) Veri transferi ve Güvenlik (DTO Pattern)

Ne yaptım? = Veritabanı tablosunu (Expense) olduğu gibi dışarı açmak yerine, araya ExpenseDto adında güvenli bir katman koydum. Parola gibi hassas verileri gizledik, tarihleri formatladım.

Teknik karşılığı = (Decoupling) (Bağımlılığı koparma) API'nın dış yüzünü (contract), veritabanı şemasından ayırdık. Select modu ile Mapping (Eşleme yaptık)

5) Değişiklik Yönetimi (Migrations)

Ne yaptım? = Modele sonradan Username ve Password ekledik. Bu değişikliği veritabanına aktarmak için migration oluşturduk. Senkronizasyon bozulunca veritabanını sıfırlayarak (RESET) düzelttim.

Teknik Karşılığı = schema migration = Veritabanı yapısının versiyonlanması ve yönetilmesi.

Sistemin Çalışma Akışı

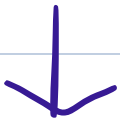
Şuan dotnet run dediğimizde arka planda şu akış gerçekleşiyor

1) İstek (Request): Kullanıcı (Swagger) /api/expenses adresine istek atar.

2) Controller = İstek GetExpenses metoduna düşer.

3)Database: Kod, Entity Framework aracılığıyla SQLite veritabanından ham veriyi (Expense) çeker.

4) Logic & Mapping: Çekilen veri filtrelenir ve DTO formatına (ExpenseDto) dönüştürülür. (Parola atılır, Tarih düzenlenir).



5) Cevap (Response): Temizlenmiş JSON verisi kullanıcıya döner.