# Adım Adım Proje Yapımı

1. Backend dosyası oluşturulur:
2. Frontend dosyası oluşturulur.

----> Bu kısıma geldiğimizide bilmemiz gerekn şeyler var:

```
- Solition: C# projelerinin en üst düzey kapsayıcısıdır. Pythonda el ile
oluşturduğumuz db/, ui/, utils/ klasörlerini, Solution yapısı otomatik ve
düzenli şekilde oluşturur

- Entity: Veritabanındaki bir tabloyu temsil eden C# sınıfıdır. EF ise ORM
sistemidir. Yani veritabanıyla C# nesneleri üzerinden çalışırız. SQL
yazmamıza gerek bile kalmaz

- ASP.NET Core: C#'da web siteleri ve REST API'ler geliştirmek için
kullanılan framework'tür. Python'daki Flask/Django gibi

- Inheritance: Diyelim ki benim bir tane sınıfım var. İçerisinde ID,
CreatedDate gibi diğer sınıflarda da kullanılacak bilgileri alıyorum
tutuyorum. Ben bu sınıfı diğer sınıflarda inherit edersem diğer sınıflarda
ID, CreatedDate gibi değerleri tekrar tanımlamama gerek kalmaz.

- Encapsulation: Diyelim ki benim bir class'da bir değişkenim var. Ben
bunun dışarıdan direkt erişilmesini istemiyorum. Ama üzerinde işlemler
yaptıktan sonra erişilebilir olmasını problem etmiyorsam encapsulation
yüntemi kullanırım. Yani Bir sınıfın iç detaylarını gizleyip sadece
dışarıya kontrollü erişim sağlamaktır.

- Interface: Bir sınıfın hangi fonksiyonları içermek zorunda olduğunu
belirleyen bir şablondur. Ama nasıl yapılacağını söylemez. Benimle çalışan
her sınıf Start() ve Stop() fonksiyonlarını içermeli, ama nasıl
çalışacağına karışmam.” gibi.

- Dependency Ingjection:
```

3. Web sayfasının backend'den veri alabilmesi için (Js'de HTTP isteği atmak için) axios kullanımaktadır. axios kurulumu yapılır. Yani frontend ve backendi bağlamış oluyoruz.
4. axios'u kurduktan sonra frontend kodunda axios'u kullanarak backend'den veri çekilir.
5. CORS durumundan dolayı backend yanıt vermemektedir bunun için portumuza izin vermemiz gerekir
6. Backend'den sahte veriler yerine artık kendi verilerimizi çekmeye başlayacağız. Bunu için ilk olarak Entity framework paketleri kurulur.
7. BlogPost modeli oluşturarak tablomuzun verilerimizi belirleriz.
8. BlogPost modelini oluşturduktan sonra bu modeli veritabanına bağlayacak olan köprüyü(DbContext) oluşturulmalıdır. DbContext aslınd C# ile MySQL arasındaki tercümandır.
9. DbContext oluşturmak için Data klasörü oluşturulup içerisinde ApplicationDbContext.cs dosyası oluşturulur ve içeriği yazılır.

10. DbContext oluşturulduktan sonra Program.cs'de yani beyin merkezinde sisteme tanıtılır.
11. Bu aşamada BlogPost modelimiz C# kodunda var ama MySQL'de henüz tabloyu oluşturmadık. Birbirini tanıtmak için Migration kullanılarak tablo oluşturulur.

> **NOT:**

DbContext, Program.cs ve Migration

> DbContext, C# kodunu SQL'e çeviren bir köprüdür. Python'da her seferinde veritabanına bağlanıp, cursor oluşturup, SQL sorgusu yazmak zorundaydın. Burada öyle değil. Sen sadece C# kodu yazıyorsun, DbContext arka planda hem bağlantıyı kuruyor, hem SQL'i yazıyor, hem de sonucu getiriyor. Mesela `db.BlogPosts.ToList()` yazdığında, DbContext bunu `SELECT * FROM BlogPosts` şeklinde SQL'e çeviriyor ve sonucu getiriyor. Sen sadece uygun fonksiyonu çağırıyorsun, geri kalanıyla uğraşmıyorsun.
>
> Peki bu DbContext'i neden Program.cs'ye ekliyoruz? Çünkü tek yerden tanımlayıp her yerden kullanmak istiyoruz. Program.cs'de bir kere tanımlıyorsun, sonra tüm projede `db.BlogPosts`, `db.Users` diye kullanıyorsun. Her controller'da tekrar tekrar tanımlamana gerek kalmıyor. Merkezi bir yönetim var, herkes aynı bağlantıyı paylaşıyor. Bu sayede hem kod tekrarı olmuyor hem de connection pooling gibi optimizasyonlar otomatik çalışıyor.
>
> Migration ise C# class'larını veritabanı tablolarına çeviren sistemdir. BlogPost diye bir class yazıyorsun, içine ID, Title, Content gibi propertyler ekliyorsun. Sonra migration oluştur diyorsun, o bakıyor class'a ve otomatik olarak `CREATE TABLE BlogPosts` SQL komutunu yazıyor. Database update dediğinde bu SQL çalışıyor ve tablo oluşuyor. Değişiklik yapmak istediğinde, mesela ViewCount eklemek istersen, class'a property ekliyorsun, yeni migration oluşturuyorsun ve o sadece `ALTER TABLE BlogPosts ADD COLUMN ViewCount` komutunu yazıyor. Yani her seferinde tabloyu silip yeniden oluşturmuyor, sadece değişeni algılayıp ona göre SQL yazıyor. Ayrıca migration'lar versiyonlanıyor, yani geri alabiliyorsun. Hata yaptıysan veya eski haline dönmek istiyorsan önceki migration'a geri dönüyorsun. Takım çalışmasında da çok işe yarıyor çünkü herkes aynı migration'ları uyguladığında veritabanı yapısı herkeste aynı oluyor.
>
> Özetle: DbContext C# kodunu SQL'e çevirip otomatik işlem yapıyor, Program.cs'ye ekliyoruz ki her yerden kullanabilelim, Migration da class'ları tablolara çevirip versiyonluyor.

---

Bu aşamadan sonra yaptıklarımıza bakmak gerekirse:

✅ Proje kurulumu - Solution + API projesi

✅ Backend test - WeatherForecast çalışıyor

✅ Frontend kurulumu - React + Vite

✅ Integration - Frontend ↔ Backend + CORS çözümü

✅ BlogPost modeli - Veri yapısı tanımı

✅ DbContext - C# ↔ MySQL köprüsü

✅ Program.cs - DbContext kaydı

✅ Migration - MySQL tablosu oluşturma

---

11. Şimdi ise WeatherForecaast yerine BlogPost verileri ile çalışan API endpointleri yazmalıyız. Bunlara Controller da denmektedir. BlogPostController.cs dosyası oluşturulur ve yazılır.

12. Kendi verilerimiz için oluşturduğumuz endpointler frontende uygulanır