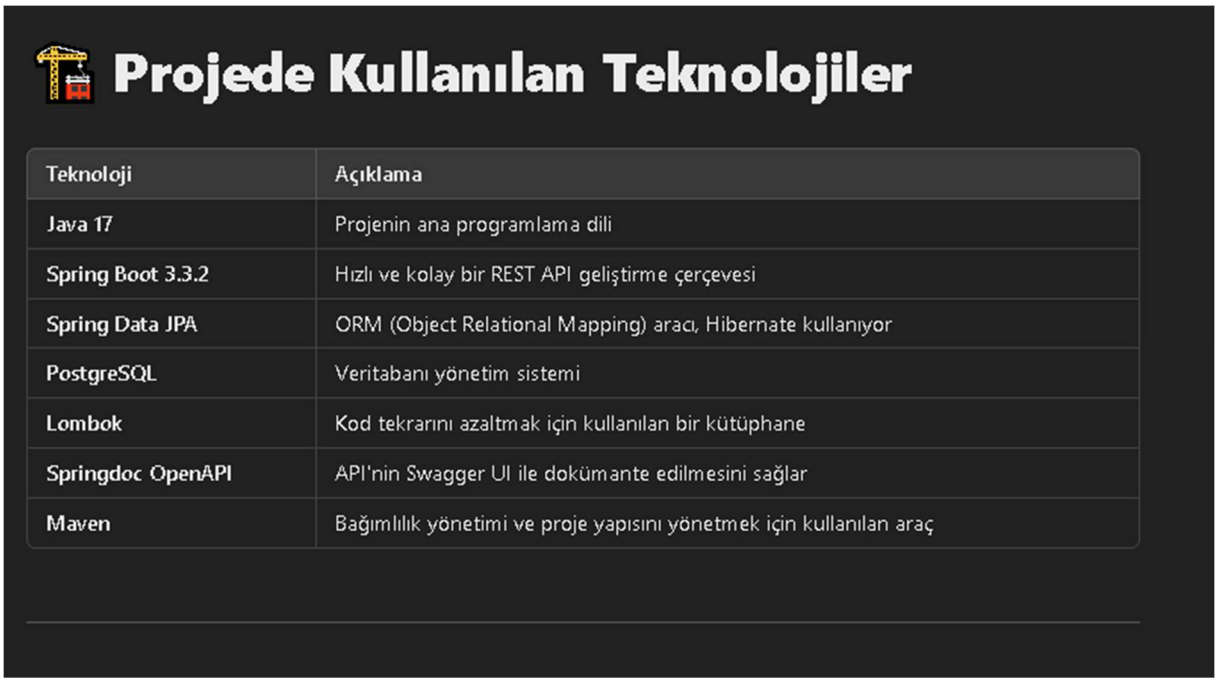
Northwind Projesi

Bu proje, **Spring Boot ile geliştirilmiş bir RESTful Web API** uygulamasıdır. Uygulama, bir veritabanından ürün (**Product**) ve kategori (**Category**) bilgilerini çekerek, bunları API üzerinden istemcilere sunar. **PostgreSQL** veritabanı kullanılıyor ve API çağrıları **Spring Web** ile yönetiliyor. **Spring Data JPA** kullanılarak veritabanı işlemleri gerçekleştirilirken, **Springdoc OpenAPI** ile API dokümantasyonu sağlanıyor.

- **Northwind**, github'ımızdaki **ReactProject** projesinin **backend veri kaynağı** olarak kullanılmıştır. Projenin nihai açıklamasına frontend'imiz olan github'ımdaki **ReactProject** README dosyasında bulabilirsiniz.

📌 Proje Genel Amacı

- **Ürünleri listelemek, eklemek ve aramak için bir API sunar.**
- **Ürünler belirli kategorilere bağlıdır ve bu ilişkiler API üzerinden görüntülenebilir.**
- **Ürünleri sayfalama (pagination) ve sıralama (sorting) işlemleri ile getirme yeteneği sunar.**
- **Spring Boot'un sunduğu pratik anotasyonlar ve bileşenler ile modüler bir mimari oluşturur.**



🛠️ Proje Bileşenleri ve Çalışma Mantığı

Aşağıda, projedeki önemli bileşenleri ve nasıl çalıştıklarını açıklayacağım.

1️⃣ Katmanlı Mimari

Proje **katmanlı bir yapı** ile geliştirilmiş. Her katmanın görevi şöyle:

1. **API Katmanı (Controller)**: HTTP isteklerini karşılar ve hizmet katmanına yönlendirir.
2. **İş Mantığı Katmanı (Service - Manager)**: İş kurallarını uygular ve veritabanı erişim katmanına yönlendirir.
3. **Veritabanı Erişim Katmanı (Repository - DAO)**: JPA ve Hibernate aracılığıyla veritabanı işlemlerini gerçekleştirir.
4. **Entity Katmanı (Model - DTO)**: Veritabanı tablolarını temsil eden sınıflardır.

2️⃣ Ana Bileşenler

Aşağıda, projenin önemli bileşenlerini ve nasıl çalıştıklarını detaylı şekilde inceleyelim.

🔷 Entity Sınıfları (Veri Modelleri)

✅ **Product.java**

java Kopyala Düzenle

```java
@Entity
@Table(name = "products")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="product_id")
    private int id;

    @Column(name="product_name")
    private String productName;

    @Column(name="unit_price")
    private double unitPrice;

    @Column(name="units_in_stock")
    private int unitsInStock;

    @ManyToOne()
    @JoinColumn(name="category_id") // Kategori ile ilişkili
    private Category category;
}
```

✅ **Bu sınıf, `products` tablosunu temsil eder ve ürün verilerini içerir.**

- `@Entity` → Veritabanı tablosunu temsil ettiğini belirtir.
- `@ManyToOne` → Her ürün bir kategoriye ait olabilir.

✅ **Category.java**

java Kopyala Düzenle

```java
@Entity
@Table(name = "categories")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="category_id")
    private int id;

    @Column(name="category_name")
    private String categoryName;

    @OneToMany(mappedBy="category")
    private List<Product> products;
}
```

✅ **Bu sınıf `categories` tablosunu temsil eder ve kategorileri içerir.**

- `@OneToMany(mappedBy="category")` → Bir kategori birden fazla ürüne sahip olabilir.

🔷 DAO (Data Access Layer - Veritabanı Erişimi)

✅ **ProductDao.java**

java Kopyala Düzenle

```java
public interface ProductDao extends JpaRepository<Product, Integer> {
    Product getByProductName(String productName);
    Product getByProductNameAndCategoryId(String productName, int categoryId);
    List<Product> getByProductNameContains(String productName);
}
```

✅ **Bu sınıf veritabanına erişimi sağlar.**

- `JpaRepository<Product, Integer>` sayesinde temel CRUD işlemleri otomatik olarak eklenmiştir.
- `getByProductName(String productName)` → Belirtilen isimdeki ürünü döndürür.
- `getByProductNameAndCategoryId(String productName, int categoryId)` → Ürün ismine ve kategoriye göre arama yapar.

🔷 Service (İş Mantığı Katmanı)

✅ ProductService.java

java — Kopyala — Düzenle

```java
public interface ProductService {
    DataResult<List<Product>> getAll();
    Result add(Product product);
    DataResult<Product> getByProductName(String productName);
}
```

✅ Bu bir arayüzdür ve iş mantığını belirler.

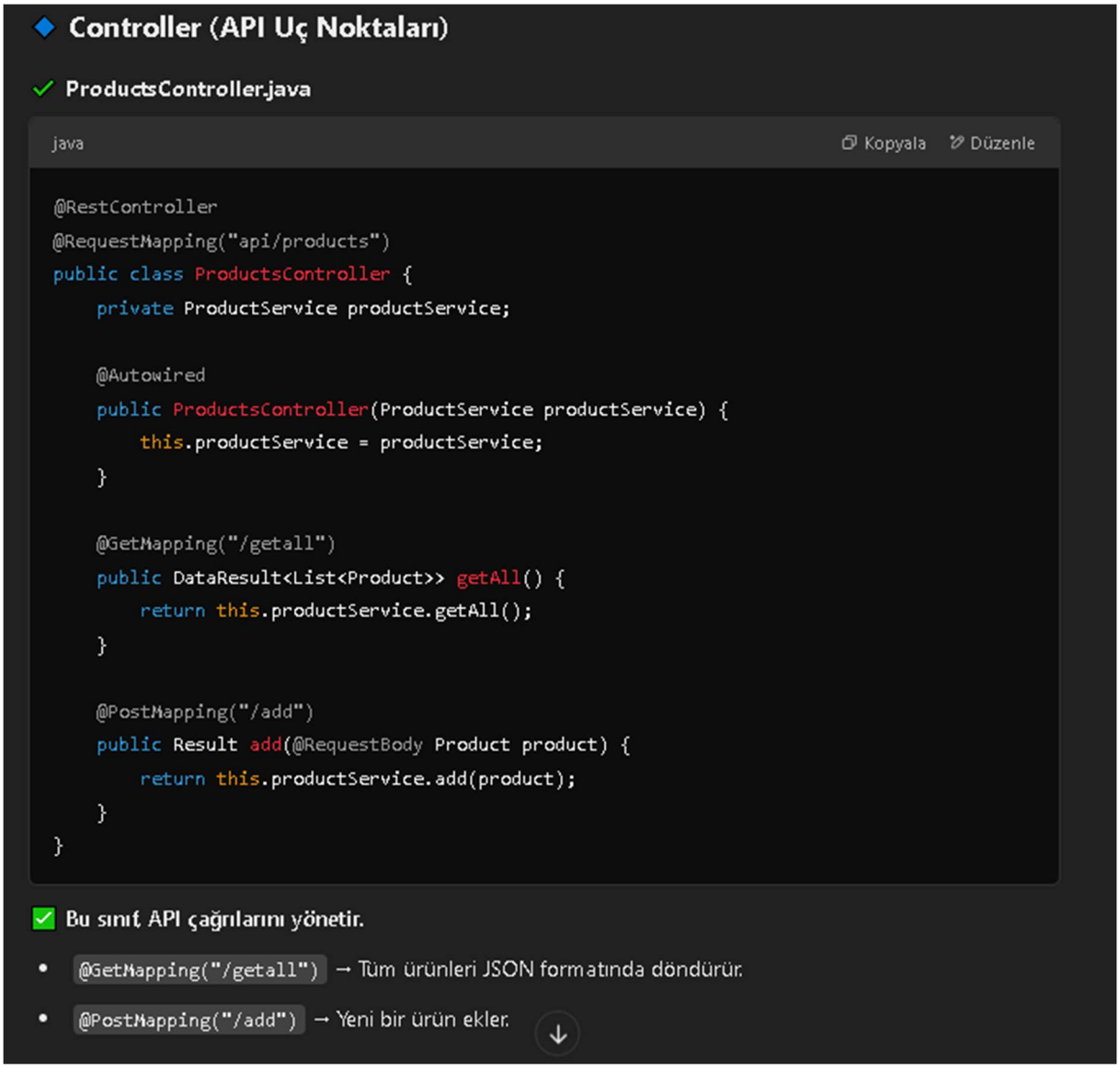
✅ ProductManager.java

java — Kopyala — Düzenle

```java
@Service
public class ProductManager implements ProductService {
    private ProductDao productDao;

    @Autowired
    public ProductManager(ProductDao productDao) {
        this.productDao = productDao;
    }

    @Override
    public DataResult<List<Product>> getAll() {
        return new SuccessDataResult<>(this.productDao.findAll(), "Ürünler listelendi");
    }

    @Override
    public Result add(Product product) {
        this.productDao.save(product);
        return new SuccessResult("Ürün eklendi");
    }
}
```

✅ Bu sınıf `ProductService` arayüzünü uygular ve iş mantığını yönetir.

- `getAll()` → Tüm ürünleri listeler.
- `add(Product product)` → Yeni bir ürün ekler.



Çalışma Mantığı

1. **İstemci (Postman, Frontend, Mobil Uygulama) bir HTTP isteği yapar.**
   - `GET /api/products/getall` → Tüm ürünleri getir.
   - `POST /api/products/add` → Yeni ürün ekle.
2. **Spring Boot, isteği `ProductsController`'a yönlendirir.**
3. **`ProductsController`, isteği `ProductService` üzerinden `ProductManager` sınıfına yollar.**
4. **`ProductManager`, `ProductDao` aracılığıyla veritabanına bağlanır ve işlemi gerçekleştirir.**
5. **Sonuç JSON formatında istemciye geri döner.**

🎯 Sonuç

Bu proje, **Spring Boot ile bir REST API uygulaması** geliştirmeyi amaçlayan, ürün ve kategori bilgilerini yöneten bir sistemdir. **Spring Data JPA, PostgreSQL, Lombok, OpenAPI gibi modern teknolojiler** kullanılarak, ölçeklenebilir bir yapı oluşturulmuştur.

Bir sonraki adımda React gibi bir frontend bağlanarak, sistem bir e-ticaret uygulamasına dönüştürülebilir. 🚀