Reşad Tayyip ÇİÇEK – Enoca Backend Document

Projede Kullandıklarım

- Spring Boot DevTools
- Spring Data JPA
- Spring Web
- Lombok
- Swagger UI
- Postgre SQL

Projenin Çalıştırılması

- Projede Java SE 11. Sürüm kullanılmıştır. Java SE 11.Sürüm aşağıdaki drive adresinden indirilip, kurulmalıdır. Eğer başka java sürümleri varsa proje çalıştırılırken 11. sürümün seçili olmasına dikkat edilmelidir.

https://drive.google.com/drive/folders/1rOxHcjBht98b7yYihKOApX2c5VPHtWzK?usp=sharing

- Projemizde kullanılan IDE Eclipse'dir. Eclipse kullanılacaksa lombok kurulmalıdır.
- Projede kullanılan ilişkisel veritabanı PostgreSQL'dir. Projenin PostgreSQL' e bağlanması için PostgreSQL şifremizin "test" olması gerekmektedir. Eğer PostgreSQL şifreniz değişik bir şifre ise projeyi çalıştırmadan önce IDE üzerinden projenin resources/application.properties dosyasından veritabanı için yazılmış olan "test" şifresini kendi PostgreSQL şifreniz ile değiştirebilirsiniz. PostgreSQL'e giriş yaptıktan sonra Enoca adında yeni bir veritabanı oluşturmanız yeterlidir. Tablolar proje çalıştırıldıktan sonra veritabanında oluşacaktır.
- Projeyi **https://github.com/RTayyipC52/Enoca-Backend** adresinden Git clone ile lokalinize klonlayıp, projeyi IDE'niz ile çalıştırın.
- Proje çalıştığında https://localhost:8080/swagger-ui.html adresi üzerinden projenin Backend'inin çalışan hali görünür.

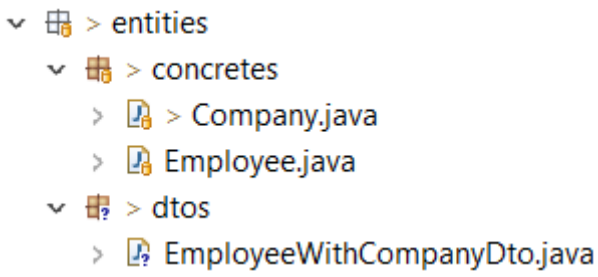
Backend Katmanlı Mimari:

- **Entities**: Veritabanındaki tabloların karşılığı olan uygulamamız içerisinde ihtiyaç duyduğumuz özellik tutan nesnelerin yazıldığı katmandır.

Entities katmanında veritabanımızdaki tabloların karşılığı olan Company ve Employee sınıflarını oluşturdum. Bu sınıfların alanlarını yazdım. Lombok kullanarak sınıfların getter, setter ve constructorlarını oluşturdum.

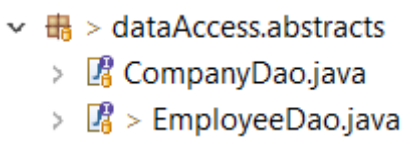
Lombok anotasyonları : @Data anotasyonu sınıfın üstüne eklenen bir anotasyondur. Sınıf içindeki tüm field'ların getter ve setter metodlarını oluşturur. **@AllArgsConstructor** anotasyonu sınıfın constructor'ını üretir. Bu constructor sınıftaki tüm field'ları parametre olarak almıştır. **@NoArgsConstructor** anotasyonu sınıf için parametresiz constructor üretir.

OEREM (Object Relational Mapping) yani veritabanındaki ilişkisel tasarımı koda yansıtıp nesnel bir yapı kurdum. Company ve Employee tabloları arasındaki ilişkiyi bir şirkette birden çok çalışan olacağı için Company sınıfında ManyToOne ve Employee sınıfında ise OneToMany olarak ilişkilendirmeyi yaptım. Çalışan bilgilerini getirirken aynı zamanda şirket bilgisini de yanında getirmeyi tek objede yapmak için Dtos klasörünü oluşturdum.



- **DataAccess (dao)** : Temel sorgular vasıtasıyla veritabanına erişip veritabanında işlemler yaptığımız yerdir. Veri erişim katmanıdır.

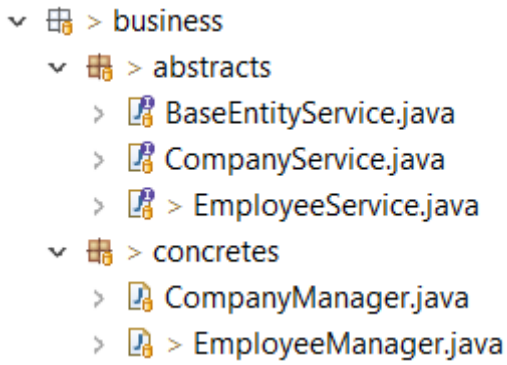
DataAccess katmanında sınıflarımız için CompanyDao ve EmployeeDao adında interfaceleri oluşturdum. Interfacelerde veritabanında işlemler yapmak için JpaRepository'i extend ederek yazdım. JpaRepository entity anotasyonuyla süslenmiş olan nesnemiz için primary key'i de verilerek yazılır. JpaRepository sayesinde DataAccess katmanında CRUD operasyonları hazır oluyor.



- **Business**: Servislerin kullanacağı operasyonların yazıldığı iş katmanı. İş kurallarının yazıldığı katmandır.

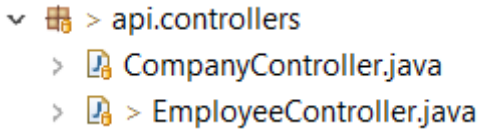
Business katmanında sınıflarımız için bir service ve bir de manager oluşturuyorum. Servislerde aynı operasyonları tüm servislere yazarak aynı kodu birden fazla yerde yamakla uğraşmamak için bir BaseEntityService oluşturdum. CRUD operasyonlarını sadece bu serviste yazıp bir generic yapı oluşturdum. Diğer servislerde bu servisi miras

alarak methodları tüm servislerde yazmış oldum. Managerlarda da ilgili servisi miras alarak methodların içlerini doldurdum. Managerlarda Dao'lara erişmek için injection yapıp methodlarda CRUD operasyonlarını kullandım.



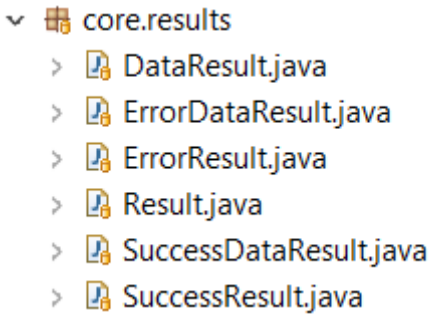
- **Api**: Back-end harici yapılarla bağlantı işlemlerinin yapıldığı paket. Projemizi dış dünyayla etkileşim sağlayabilecek alt yapıya getirdiğimiz katmandır.

Controller dış sistemlerin bizim sistemimizle iletişim kurduğu yerdir. Controllerda gelecek olan istekleri karşılayacak methodları yazdım. Burada da servislere erişmek için injection yapıp CRUD operasyonlarını getirdim.



- **Core** Katmanı: Tüm sistemimizde kullanabileceğimiz ve aynı zamanda yeni projelerimizde de tekrar kullanabileceğimiz kodları içeren katmandır.

Core katmanını bütün projelerde kullanabileceğimiz ortak kodları koymak için oluşturdum. Burada Request'lere verilecek olan Response'ları standartize ediyorum. Burada Results klasörü oluşturdum. Result sınıfında başarılı mı ve işlem sonucundaki mesaj, DataResult sınıfında yanında veriyi de gönderiyoruz. Başarılı olma durumu için SuccessResult, başarısız olma durumu için ErrorResult oluşturdum. Yanında veriyi de göndermek için başarılı olma durumunda SuccessDataResult, başarısız olma durumunda ErrorDataResult oluşturdum.



- **pom.xml**: Proje hakkında gerekli bilgileri ve konfigrasyonu içeren bir xml dosyası.