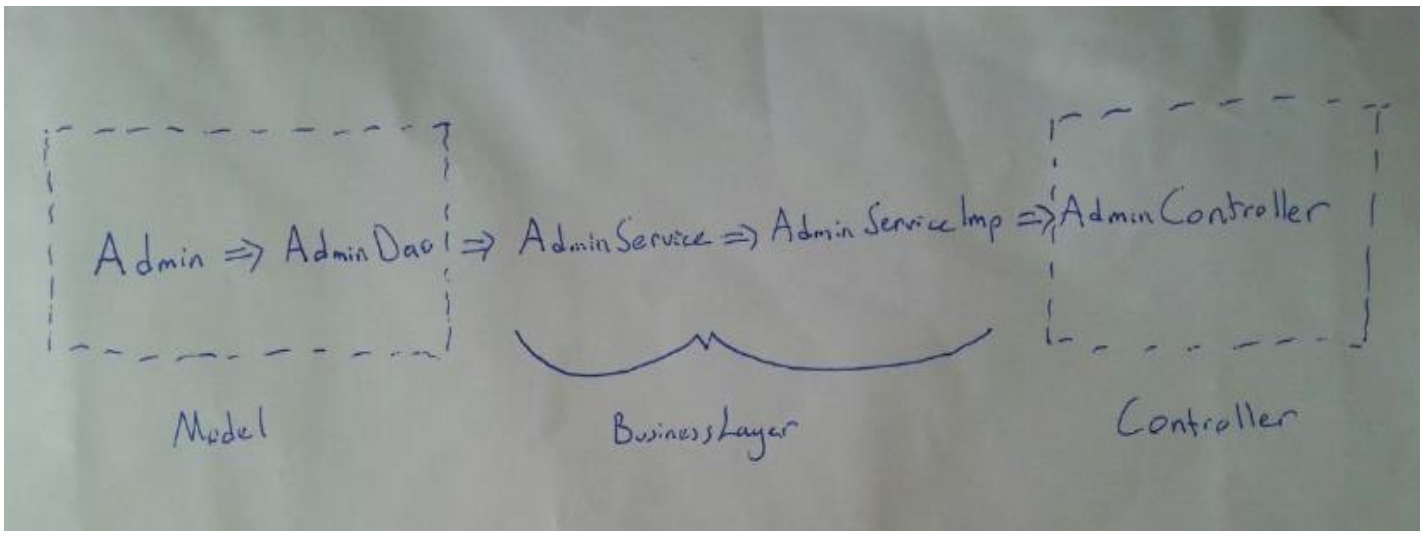

Projede kullanılan frameworkler; **Spring, Spring MVC, Spring DATA.**

Projeyi kendi bilgisayarına çektikten sonra "com.blog.river.admin" package adresine gittiğinde bu isimlerde oluşturulmuş Java dosyaları göreceksin.

MVC mimarisine göre geliştirme yapıyoruz.Fakat projemiz REST API olarak çalışacağı için bir View katmanına Java tarafında ihtiyacımız olmuyor.Çünkü geriye REST API'den gelen JSON data dönecek.

Şimdi yukarıdaki resimde **soldan sağa** olmak üzere **geliştirme** işlemimizi yapıyoruz. Fakat projenin **çalıştığı zamanki akışı**nı göz önüne alırsak bu sefer **sağdan sola** doğru bir akış olacaktır.

Örneğin; Kullanıcı bir url üzerinden istek yolladı. Bu Controller'a düştü.Controller, Business Layer'a yapılması gereken işi söyledi. Business Layer işi yapması için gerekli olan data'yı Model katmanından aldı ve işledi.

Bu gördüğün yapı aslında piyasadaki projelerin neredeyse hepsinin aslında kullandığı yapı. Tek tek açıklayayım.

Model: Veritabanı işlemlerinin döndüğü katman oluyor.

- **Admin**: Veritabanındaki tablomuzun ismi.Bu sınıf @Entity anotasyonu ile işaretlenmiş.Bu da bunun bir tablo olacağını söylüyor.
- **AdminDao**: Dao'nun açılımı Data Access Object'dir(Design Pattern). Yani o data modeline erişim için kullanacağımız ara bir katmandır. Veritabanı üzerinde yapacağımız kayıt,silme,listeleme vs. gibi işlemler için bu ara katmanı kullanıyoruz. İnce bir detay olarak bu Interface *CrudRepository<Admin, Long>* Interface'ini extends ediyor. Bu Interface Spring Data framework'ü içinde bulunan bizim için tekrarlayan işlemleri(save,delete,find vs.) bizim yerimize çözen bir yardımcı sınıf. Bunu kullanarak Bu AdminDao sınıfımızın bir implementasyonunu kendimiz yaratmıyoruz. Spring Data bizim yerimize kendi nesnesini çalışma zamanında AdminDao Interface'imize atıyor.

Business Layer: Türkçe anlamı **İş Katmanı** olan bu katmanda Model ile Controller katmanlarımız arasındaki iş yapılır.İşten kasıt nedir? Her zaman Model katmanından gelen verilerin doğrudan cevap olarak döndürülmesi gerekmeyebilir. Bu akış içerisinde veriler üzerinde bazı değişiklikler yapılması veya bazı eklemeler yapılması gerekebilir. Örneğin; Article kayıt işlemi yapılırken bir de yazar için bir sayaç tutulması gerektiğinde yani ek bir iş olduğunda işte bu katmanda işlerimizi hallediyoruz.

- **AdminService**: Bu iş yapacak sınıfımızın Interface hali. Böylece ilerde farklı bir çözüm(implementation) kullanmak istediğimizde sistem darbe almadan onu kullanabiliriz.
- **AdminServiceImp**: Yarattığımız Interface için yazdığımız implementasyon. Admin ile ilgili işler bu sınıf üzerinden yürüyor.

Controller: Kullanıcıdan URL üzerinden gelen isteklerin yakalandığı ve ilgili yerler ile haberleşen katmandır.

- **AdminController**: Admin işlemlerimiz için kullandığımız Controller sınıfıdır.Bu sınıf üzerinden istekler cevap bulur.

Ek bilgiler;

Anotasyon nedir?
Anotasyonlar (Annotations) Java 5'den beri Java ortamında kullanılan bileşenlerdir. Notasyonlar genel olarak bir bileşene özellik katma veya konfigürasyon amaçlı olarak kullanılmaktadır. Yani bunu kullandığımızda arka tarafta bizim için yapılması gereken bir iş yükünü arka tarafda kendisi yapar. Eğer Anotasyon Spring içerisinden geliyorsa bunu Spring yapar.Hibernate içinden geliyorsa Hibernate yapar.

Projede kullanılan Anotasyonlar ve anlamları;

- **@Entity**: Tanımlandığı sınıfın bir varlık-model(entity) olduğunu belirtir.Böylece o sınıf bir data modeli olarak anlaşılır.
- **@Table(name = "admin")**: Bu sınıfın veritabanında bir tabloya denk geldiğini ve isminin veritabanında "admin" olarak kullandıldığını veya kullanılacağını belirtir.
- **@Id**: Sınıfı bir tablo olarak algılamaya başlarsak; değişkenlerde birer kolon olacaktır. Id kolonumuzu tanıtmak için bu anotasyonu kullanıyoruz.
- **@GeneratedValue(strategy = GenerationType.AUTO)**: id değerinin auto increment yani otomatik kendiliğinden artmasını bu şekilde söylüyoruz. Bu sayede nesnenin özelliklerini atarken id'ye hiç dokunmuyoruz.
- **@NotNull**: Tanımlandığı değişkenin null olamayacağını belirtir.
- **@Controller**: Tanımlandığı sınıfın Controller görevinde olduğunu belirtir. Kullanıcıdan gelen URL istekleri bu anotasyon tanımlı sınıflar içinde aranır.
- **@RequestMapping("/create")**: Controller sınıfı içindeki metodların hangi istek doğrultusuna çalışması gerektiğini belirtir. (Request Mapping = İstek Eşleştirme)
- **@Service**: Bu sınıfın bir Interface'i implement ettiğini söyler.Böylece o Interface'in üzerinde tanımlanan @Autowired anotasyonu bu nesneyi Interface'e otomatik atar.
- **@Autowired**: Tanımlandığı Interface değişkenine çalışma zamanında implementasyonunu atar.(Dependency Injection işleminin pratikte uygulandığı Anotasyon budur.)