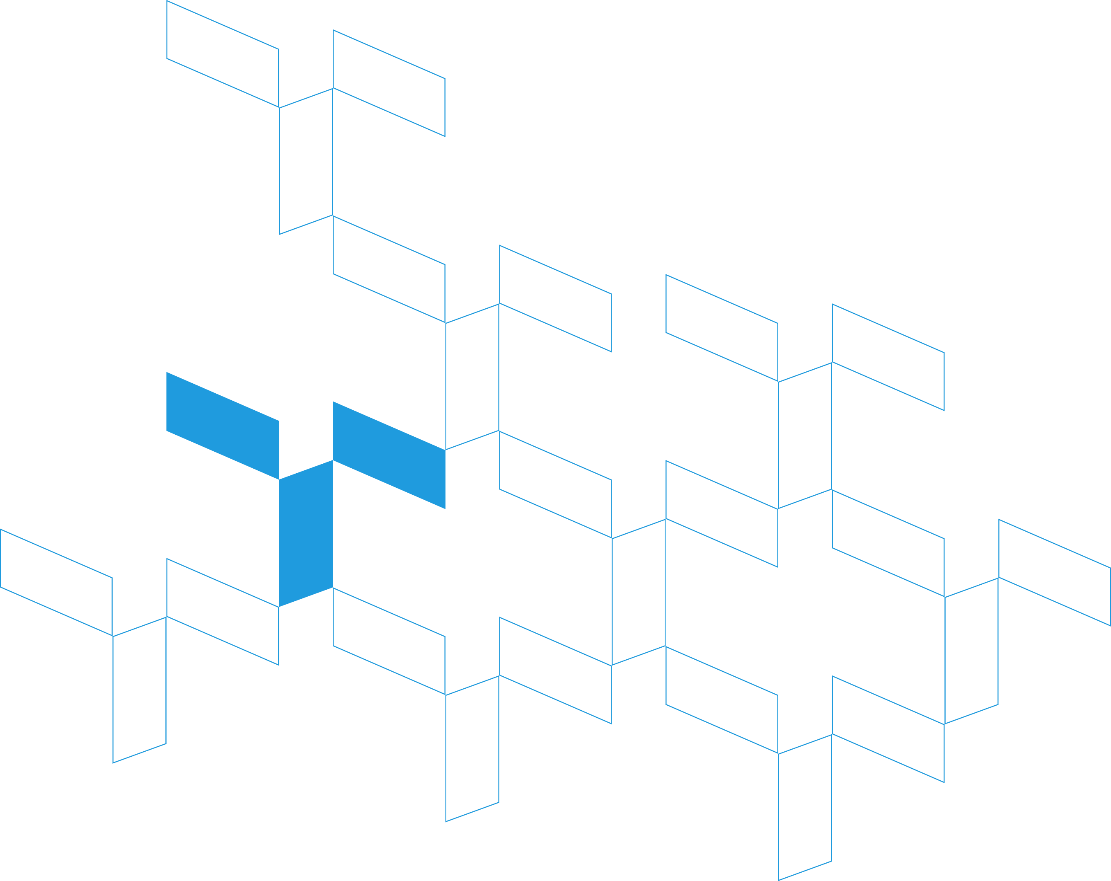
****

**Yazılım Tasarım Prensipleri ve Uygulanması Gereken Esaslar**



**Tarih: 22.05.2018**

**Versiyon: v1.0**

**İÇİNDEKİLER**

[1 Mimaride Temel Tasarım Uygunlukları Ve Teknolojileri 4](#_Toc514848104)

[1.1 Yazılım Temel Tasarım Uygunlukları 4](#_Toc514848105)

[1.2 İç Kod Kalitesi 5](#_Toc514848106)

[1.2.1 Tasarım 5](#_Toc514848107)

[1.2.1.1 İç isimlendirme Ve Uygunluk 5](#_Toc514848108)

[1.2.1.2 Kod İçi Dokümantasyon 6](#_Toc514848109)

[1.2.1.3 Paket Yapısı Ve Tasarım 6](#_Toc514848110)

[1.2.2 Test 7](#_Toc514848111)

[1.2.2.1 Test Kapsama Alanı (Test Coverage) 7](#_Toc514848112)

[1.2.2.2 Birim Testi 7](#_Toc514848113)

[1.2.2.3 Entegrasyon Testi 8](#_Toc514848114)

[1.3 Katmanlarda Tasarım Uygunlukları 8](#_Toc514848115)

[1.3.1 Solid Tasarım Prensipleri 8](#_Toc514848116)

[1.3.2 Paket Bazlı Tasarım Prensipleri 8](#_Toc514848117)

[1.4 Veri Erişim Katmanında Tasarım Uygunlukları 9](#_Toc514848118)

[1.4.1 ORM Yaklaşımları Ve Destekleyici Unsurlar 9](#_Toc514848119)

[1.5 Sunum Katmanında Tasarım Uygunlukları 10](#_Toc514848120)

[1.5.1 Sayfa Tasarım Ve Template Uygunlukları 10](#_Toc514848121)

[1.6 Dış Kaynak Kullanım Kısıtları 11](#_Toc514848122)

[1.7 Dikey Katman Tasarım Uygunlukları 11](#_Toc514848123)

[1.7.1 Yardımcı Unsurlar (Helpers) 11](#_Toc514848124)

[1.7.2 Hata Yönetimi (Exception Handlers) 12](#_Toc514848125)

[1.7.3 Onaylayıcılar (Validations) 12](#_Toc514848126)

[1.7.4 Kayıt Altına Alma (Loging) 12](#_Toc514848127)

[1.7.5 Ön Bellekleme 12](#_Toc514848128)

[2 MVC Yaklaşımı ve Spring Framework 12](#_Toc514848129)

[3 Yazılım Geliştirme Teknolojileri Ve Uygunlukları 13](#_Toc514848130)

[3.1 Uygulanan Teknolojiler 13](#_Toc514848131)

[3.2 Modüler Tasarım Uygunlukları 14](#_Toc514848132)

[3.2.1 Modül Bazlı Servis Tasarımı 14](#_Toc514848133)

[3.3 Entegrasyon Testi 14](#_Toc514848134)

[3.4 Birim Testi 14](#_Toc514848135)

[3.5 Log Araçları 14](#_Toc514848136)

[3.6 Statik Kod Analizi 14](#_Toc514848137)

[3.7 Raporlama 15](#_Toc514848138)

[3.8 Servis Katmanı Ve Tasarımı 15](#_Toc514848139)

[3.9 Zamanlanmış Görev Kapsamı 15](#_Toc514848140)

[3.10 Konteyner Uygunlukları 15](#_Toc514848141)

[3.11 Mesaj Kuyruğu Sistemi 15](#_Toc514848142)

[3.12 Bağımlılıkların Yönetimi 16](#_Toc514848143)

[3.13 Dinamik İş Süreçleri Ve Kuralları 16](#_Toc514848144)

[3.13.1 BPM 16](#_Toc514848145)

**Şekiller**

[Şekil 1Modüller arası iletişim gösterimi 4](#_Toc514848146)

[Şekil 2Modül tasarımı yaklaşımı 5](#_Toc514848147)

[Şekil 3Temel dosya düzeni 7](#_Toc514848148)

[Şekil 4Test paket yapısı 7](#_Toc514848149)

[Şekil 5İsimlendirme ve loglama 12](#_Toc514848150)

[Şekil 6Docker konteyner yapısı 15](#_Toc514848151)

[Şekil 7Basitçe çalışma mimarisi 16](#_Toc514848152)

[Şekil 8Maven sunucusunun çalışma şeması 16](#_Toc514848153)

# Mimaride Temel Tasarım Uygunlukları Ve Teknolojileri

## Yazılım Temel Tasarım Uygunlukları

Yazılım tasarımı gerçekleştirirken “iç kod kalitesi” ve “dış kod kalitesi” olmak üzere iki unsur ile üzerinde durmaktayız. Bu iki unsur da yazılımın gelişim süreçlerindeki temel adımları betimler ve tasarım aşamasından kodlama aşamasına uygun şekilde ilerlenmesinde destek olmaktadır. Yazılımda modülerliği, bakım kolaylığı, tekrar kullanma ve esneklik gibi öğeleri sağlayabilmek için yazılımda minimumda MVC, Business ve DAO katmanı bulunmalıdır ve bu katmanlar arası veri transferi ağır ve hantal veri tabanı entity objeleri yerine DTO(Data Transfer Object) diye adlandırdığımız ihtiyaca göre uygun nesnelerle taşınmalıdır.

Şekil 1 de ki görselde anlatıldığı gibi yazılım modülleri arasındaki ilişki loose couple prensibine uymalı gevşek bağlı olmalıdır. Yazılım modülleri arasında veri transferi rest servislerle olmalıdır. Doğrudan bir modül başka modülün sınıfını oluşturarak yada inject ederek erişmemelidir.

DTO

ENTITY

Object Mapper

**RestEndPoint**

Security Filter

Controller

Modul

Controller

Controller

DAO

Controller

Controller

Business

Modul

**RestEndPoint**

Security Filter

DTO

ENTITY

Object Mapper

**POST**

**GET**

**PUT**

**DELETE**

**HTTPS**

Controller

Controller

Controller

Controller

Controller

Controller

Business

Controller

DAO

Şekil Modüller arası iletişim gösterimi

Şekil 2’de anlatıldığı gibi her modülün kendi pom yapısı olmalı ve genel bir parent pom yapısından kalıtılmalıdır. Bu sayede yazılımın herhangi bir modülü diğer modüllere ihtiyaç duyulmadan derlenebilmelidir.

M1Personel

**Maven Modul**

Bağımlıklıklar

Restful Endpoinds

M2Rapor

**Maven Modul**

Bağımlıklıklar

Restful Endpoinds

Şekil Modül tasarımı yaklaşımı

## İç Kod Kalitesi

### Tasarım

#### İç isimlendirme Ve Uygunluk

Aşağıdaki isimlendirme kalıpları listelenmiştir. Her isimlendirme en fazla 30 karakterden oluşabilir. Amacına uygun isim verilerek barındırdığı bilgi hakkında geliştiricilere izlenim vermek durumundadır. **Uygulama içinde tanımlanacak olan tüm değişken, sınıf, paket, interface gibi tüm varlıklar Türkçe tanımlanacaktır.** (-üğşçöİı yerine –ugscoIi karakterleri kullanılmalıdır.)

**Yazılımda İngilizce olacak kısımlar;**

1. Yazılım ana dosya yapısı src/main/resources, src/main/webapp.
2. Konfigurasyon yada tanım dosyaları spring-security-context.xml, applicationContext.xml gibi
3. Teknik isimlendirmeler OrnekDAO, OrnekService, OrnekFilter,OrnekHandler,OrnekImpl

|  |  |  |
| --- | --- | --- |
| Tip | Model | Örnek |
| Değişkenler | Camel Case | ornekDegisken |
| Diziler | Camel Case ve Çoğul | ornekVeriler |
| Method Parametresi | Camel Case | ornekParametre |
| Methodlar | Camel Case | ornekMethod |
| Sınıf | Pascal Case | OrnekSinif |
| Interface | Pascal Case | Ornek |
| Enum | Pascal Case + UPPER CASE | OrnekTipler.TIP1 |
| Sabitler | UPPER CASE | ORNEK\_SABIT |
| Annotation | Pascal Case | @OrnekNot |
| Xml File | Lower case | ornekdosya.xml  ornek-dosya.xml |
| XHTML | Camel Case | ornek.xhtml, ornekPersonel.xhtml |
| Property File | Lower case | ornekdosya.properties |
| Dao | Pascal Case + DAO | OrnekSinifDAO |
| Service (Sınıf) | Pascal Case + Impl | OrnekSinifImpl |
| Service (Arayüz) | Pascal Case | OrnekService |
| Entity (Varlık) | Pascal Case | OrnekSinif |
| Tablo İsimleri | Lower Case | ornek\_ozellikler\_tablosu |
| Controller | Pascal Case | OrnekController |

#### Kod İçi Dokümantasyon

Yazılım varlıklarının ne amaç ile oluşturulduklarını tanımlayan text alanlardır. Bu alanlar tanımlamanın özet bilgilerini içermektedir. Açıklama kısmında bulunan metinlerin düzgün görüntülenmesi için gerekli format biçimleri ve birtakım taglar bulunmaktadır. Aşağıda bu liste belirtilmiştir.

|  |  |  |
| --- | --- | --- |
| Tip | Yazım Biçimi | Örnek |
| Sınıf,Enum | Sınıfın en üst boş alanına yazılır. |  |
| Interface | Interfacein en üst kısmına yazılır. |  |
| Değişkenler | Değişkenin üst bitişik satırına yazım yapılır. |  |
| Methodlar 1 | Methodun üst bitişik satırına yazım yapılır. |  |
| Methodlar 2 | Methodun üst bitişik satırına yazım yapılır. |  |

#### Paket Yapısı Ve Tasarım

Projede kullanılacak olan paket yapısı ile iç varlıkların düzeni sağlanmaktadır. Bu sayede aynı görev niteliği taşıyan sınıflar ve ara-yüzler gibi yazılım unsurlarının uygun şekilde kümelenmesini sağlamaktayız. Örnek olarak aşağıdaki şekil 3’te ki Maven standart dosya düzeni kullanılabilir.

|  |  |
| --- | --- |
| Model | Tanım |
| src/main/java | Uygulama/Kütüphane kodları |
| src/main/resources | Uygulama/Kütüphane kaynakları |
| src/main/filters | Filter kod dosyaları |
| src/main/webapp | Web uygulaması kaynakları |
| src/test/java | Test kodları |
| src/test/resources | Test kaynakları |
| src/test/filters | Test kaynak filtreleri |
| src/it | Entegrasyon testleri için (Öncelikle eklentiler için) |
| src/assembly | Entegre etme tanımlayıcıları |
| src/site | Site |
| LICENSE.txt | Projenin lisans bilgileri |
| NOTICE.txt | Proje bağımlılıklarının ihtiyaç duyacağı çalışma ortamı hakkında bilgilendirmeler. |
| README.txt | Proje için gerekli ön doküman. |

Şekil Temel dosya düzeni

### Test

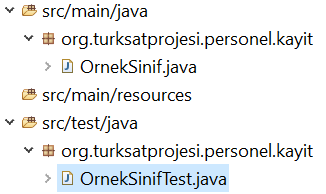
Test-case leri yazılmış ve kısa tutulmuş kod bloklarından meydana gelmiş bir uygulama anlaması kolay ve geliştirme sürecine dahil olma evresi kısadır. Fakat test yapılanmasının yeterli olmadığı projelerde temiz kod yazım süreci önem taşımamaktadır çünkü test olmadan uygulamanın yeniden revize edilme süreci katlanarak artmaktadır.

#### Test Kapsama Alanı (Test Coverage)

JUnit testleri ve gerekli araçlar yardımıyla kodun hangi satırlarının işlem gördüğünü, hangi satırların hiç kullanılmadığını ve ölü olduğunu tespit edebiliriz. Bu işleme testin kapsama alanı tespiti denir. Test kapsama alanı ne kadar geniş olursa, programın o kadar büyük bölümü test edilmiş demektir. Test kapsama alanının ölçümü, doğru testlerin yapıldığı anlamına gelmez, ama en azından kodun hangi bölümlerinin işlem gördüğünü tespit etmiş oluruz ve gerekli durumlarda test sayısını artırarak test kapsama alanını genişletebiliriz. Yazılımda Business ve DAO katmanının testi %100 e yakın olmalıdır ve genel test kapsam alanı en az %30 olması gerekmektedir.

#### Birim Testi

Yazılım en küçük birimlerinin test edilmesinden oluşmaktadır. Her birim sınıf ve method öğelerini temsil eder. Yazılım uzmanı tarafından gerçekleştirilir ve yazılım isterlerinin (SRS) kod karşılıklarının functional ve non-functional kabiliyetlerinin teyidine odaklanır. Methodların testi geçmesi yazılımın diğer modüller ile uyumlu çalışacağı garantisini vermez. Entegrasyon testleri ile gerekli süreci kontrole devam edilmelidir. Birim test oluşturma aşamasının paket yapısı ve isimlendirme örneği şekil 4’deki gibidir.



Şekil Test paket yapısı

Yazılım geliştirme süreçleri TDD (Test Driven Development) odaklı ilerlemelidir. Agile yazılım geliştirme yaklaşımı ve onun merkezinde yer alan TDD tekniği, hızlı, verimli ve değişim odaklı kaliteli uygulama geliştirme imkanı vermektedir.

**Temel prensipler**:

* İş akışı açısından kritik öneme sahip tüm methodların birim testleri yazılmalıdır.
* Test sınıfının ismi, test edilecek sınıfın isminin sonuna Test ifadesi eklenerek oluşturulmalıdır.
* İlgili birim testi, ilgili sınıfın paket ismi ile aynı isme sahip olmalı ama test dosyasının altında yer almalıdır.
* Her method yalnız bir test methodu ile test edilir.
* Her seferinde yalnızca bir senaryo test edilir.
* Behaviour Driven Development prensibi ile yazılmalıdır: Given-When-Then (Gerekli ortamın hazırlanması-Senaryonun Tetiklenmesi- Nasıl davranış sergilendiğinin kontrol edilmesi)

#### Entegrasyon Testi

Entegrasyon testleri tasarlanan modüllerin birbirleri ile olan ortak iletişim kabiliyetlerinin hatasız olduklarını teyit etmektir. Test işlemi için Top-Down veya Bottom-Up entegrasyon testleri ile gerekli kontroller yapılıp uygun şekilde onaylanmalıdır.

## Katmanlarda Tasarım Uygunlukları

### Solid Tasarım Prensipleri

Kodlama ve geliştirme aşamasında yazılan kodların güvenilirliğini ve esnek yapı gösterme eğilimin olup olmadığının incelenmesi önemli bir unsurdur. Bu süreçte birim testlerinin yapılandırılması ve bu sürece sadık kalınması önemlidir. Bir takım prensipler ile bu süreçte kodlayıcının büyüyen sistemi kontrol altında tutmasını sağlayan yaklaşımlar ele alınmalıdır. Bu yaklaşımların başında SOLID tasarım modeli gelmektedir.

* **SRP**: Single Responsibility Principle: Her yazılım biriminin tek bir sorumluluğu olmalıdır.
* **OCP**: Open Closed Principle: Çevik süreçlerde özellikle değinilen bir yaklaşım modeli olarak görülür. Yazılan modüllerin değişime mümkün olduğunca kapalı ama gelişime müsait olmalıdır. Yani modüller yeni davranış biçimlerini sergileyebilecek şekilde genişletilebilmelidir.
* **LSP**: Liskov Substitution Principle: Üst sınıf nesnelerinin kullanıldığı metotlar içinde alt sınıftan olan nesneler aynı davranışı sergilemek zorundadır, çünkü oluşturulan metotlar üst sınıf davranışları örnek alınarak programlanmıştır. Alt sınıflarda meydana gelen davranış değişiklikleri, bu metotların hatalı çalışmasına sebep verebilir.
* **ISP**: Interface Segregation Principle: Herşeyi ihtiva eden interface yapıları yerine belli bir işlemi yapan interfaceler oluşturulmalıdır.
* **DIP**: Dependency Inversion Principle: Bu prensibe göre somut sınıflara olan bağımlılıklar soyut sınıflar ve interface yapıları kullanılarak ortadan kaldırılmalıdır, çünkü somut sınıflar sık sık değişikliğe uğrarlar ve bu sınıflara bağımlı olan sınıflarında yapısal değişikliğe uğramalarına sebep olurlar.

**Ek Bilgi**: Yazılım geliştirirken özellikle basit yaklaşım öne çıkmaktadır.

* **KISS**: Keep It Stupid Simple : Her method mümkün olduğunca kısa tutulmalı ve anlaşılması kolay yazım yöntemine gidilmelidir. Bu yaklaşım SRP prensibine doğrudan atıfta bulunur.
* **DRY**: Don’t Repeat Yourself: Yazılmış olan methodlar incelenmeli ve yazılması istenen amaca uygun method var ise yeniden efor sarf edip hem zaman kaybı yaşanmamalı hem de kod karmaşasına yol açılmamalıdır.
* **Bağımlılıklar Soyut Sınıflara Olmalı**: Gerçek nesneler değişime açıktır ve isterlere bağlı olarak revize edilebilirler. Bu gibi süreçlerden minimum etkilenmek için bağımlıkları soyut sınıflara doğru yönlendirmeliyiz.

### Paket Bazlı Tasarım Prensipleri

Paketler arası bağımlılıklar ve genel prensipleri kapsamaktadır. Paketler arası bağımlılıklar doğal bir süreçtir lakin bu bağımlılıkları belirli bir sistematiğe oturtmak gereklidir.

* **REP** : Reuse-Release Equivalence Principle: Bir modülde uygun olan paket aynı türden ve nitelikten oluşan süreçleri barındırmalıdır.
* **CRP**: Common Reuse Principle: Bu prensip hangi sınıfların aynı paket te bulunması gerektiğine odaklanır. Birbirlerine bağımlılıkları olmayan, birbirlerini kullanmayan sınıfların aynı paket içinde olması sakıncalıdır.
* **CCP**: Common Closure Principle: Yazılım isterlerine bağlı olarak sistemin büyük bir bölümünü etkilemesinin önüne geçmek için aynı sebepten ötürü değişikliğe uğrayabilecek sınıfların aynı paket hiyerarşisi içinde yer alması gereklidir.
* **SDP**: Stable Dependency Principle: SDP paketler arası bağımlılık yönünün stabil paketlere doğru olması gerektiğini söyler. Stabil paketler yapı itibariyle diğer paketlere oranla daha az değişikliğe uğrarlar. Bir paketin stabillik oranını bağımlılık yönü tayin eder. Stabil olmayan bir pakete bağımlılık duyan bir paket doğal olarak değişikliklere maruz kalacaktır ve stabil kalamayacaktır.

## Veri Erişim Katmanında Tasarım Uygunlukları

### ORM Yaklaşımları Ve Destekleyici Unsurlar

Veri erişim katmanında bulunan varlıklar gerekli paket uygunluk yapısına dikkat edilerek tasarım yapılmalıdır. Veri katmanında kullanılacak olan temel prensipler şöyledir;

* Entity’ler BaseEntity yapısı barındırmalıdır.
* BaseEntity: Ortak bileşenleri içerir. id, olusturulma\_tarihi, guncellenme\_tarihi, aktif\_mi, silindi\_mi,versiyon gibi alanlar base yapı üzerinde konumlandırılmalıdır.
* Her entity için tanımlanan;
  + Entity Name:kullanici, kullanici\_rol, kullanici\_ayarlari
  + Foreign Key: kullanici\_id

şeklinde olmalıdır.

* Yazılım veritabanı bağımsız olmalıdır. Anlık olarak veritabanı değiştirilebilir tasarlanmalıdır.
* Veritabanına özgü Triger gibi yapılar kullanılamaz. Entity Listener desteği olarak ihtiyaç duyulan varlıklar için callback methodları kullanılmalıdır.
  + Örn: @PrePersist, @PreUpdate, @PreRemove, @PostPersist, @PostUpdate, @PostRemove notları ile betimlenerek entity süreci takip edilebilir.
* Eğer entity listener gurubu olarak tanımlanacak bir sınıf tercih edilecek ise uygun paket isimlendirmesi ile konumlandırılmalıdır.
* Veritabanı varlıklarından olan Stored Procedure, Stored Function, Function, Custom View (@Immutable kullanılabilir), Materialized View , Custom Trigger, Types, Sequences gibi veritabanına özgü varlıklar kullanılamaz.
* Framework olarak Spring Data JPA tercihi yapılacak ise QueryDSL ve CrudRepository kullanılarak gerekli sorgulamalar yapılacaktır.
* Sadece Hibernate kullanımı tercih ediliyor ise SessionFactory autowired edilerek BaseDAO sınıfları oluşturulmalıdır.
* Hibernate kullanılacak ise Criteria API ile sorgulama yapılmalıdır. Tercih edilmemekle birlikte ihtiyaç halinde hql kullanımı yapılabilir ama gerekçe belirtilmelidir.
* Bütün konfigürasyonlar sınıf içi notasyonel yapılar ile belirtilmelidir (Full Java Config). Konfigürasyona bağlı xml dosyaları minimum tutulmalıdır (İhtiyaç duyulacak dış kaynak ayar dosyaları hariç).
* Sorgular kesinlikle Native Sql Kod ile yazılmamalıdır.
* Sorgular kesinlikle veritabanına özgü komut grupları içermemelidir.
* NamedQuery kullanılmamalıdır. Esnek tasarımın önüne geçildiği için tercihler arasında bulunmamaktadır.
* Fetch Type kullanımına dikkat edilmelidir. Senaryo bazlı değişiklik gösterebilir fakat mümkün olduğunca Lazy loading kullanılmalıdır.
* Veritabanından kayıt çekerken Projection kullanarak sadece ihtiyaç duyulan verilen çekilerek performans artışı sağlanmalıdır.
* Notasyonlar değişken üzerinde tanımlanmalıdır. Getter/Setter üzerinde herhangi bir notasyon olmayacaktır.
* Veritabanı connection-pool için third-party bir api kullanımında c3p0 api yada HikariCP kullanılmalıdır.
* Performans artışı için cacheleme mekanizması kullanılmalıdır ve Hibernate kullanılıyor ise 2nd level cache olarak EHCache tercih edilmelidir.
* Full-Text-Search, Index-Based-Search, sorgulamalar ve öneriler gibi ihtiyaçlar doğrultusunda ElasticSearch kullanılacaktır.

## Sunum Katmanında Tasarım Uygunlukları

### Sayfa Tasarım Ve Template Uygunlukları

Projelerde View katmanı unsuru belirli bir standart doğrultusunda geliştirilmediği takdirde spaghetti-code üretilmesi kaçınılmazdır. Türksat tarafından belirtilen template ile gelen html, css ve js dosyalarının haricinde geliştirilecek her eklenti modül adını ve ekranı ifade eden bir klasör isimlendirmesi ile tutulacaktır. Eğer JQuery ile geliştirilmesi elzem bir eklenti var ise uygun modül adını yansıtır şekilde klasör ağacı betimlenerek tutulmalıdır. Eğer kullanılacak bir eklenti mevcut ise Türksat ile gerekli bilgi paylaşılması gereklidir. Eklentinin uygunluğu fork değerlerine, destekleyici grup niteliğine ve bug-fix süreçleri analiz edilerek karar verilecektir. Ana işlevler JSF bileşenleri tarafından yürütülecek şekilde kod geliştirilmelidir. Bu doğrultuda geliştirilecek ekranlarda birtakım standartlar yer almalıdır. Tüm modüllerce ortak olarak kullanılan css, js gibi dosyalar ortak dosyalar alanına yerleştirilecektir.

Örn: Ortak alan:

/resources/common/js/ortakaraclar.js

/resources/common/css/ortaktasarim.css

Örn: Modül bazlı alanlar:

/resources/personel/kayit/tarih.js

Şöyle ki;

* View katmanında JSF teknolojisi kullanılacaktır.
* JSF Oracle sürümü Mojarra son stabil sürüm kullanılacaktır.
* JSF Framework standartı olarak PrimeFaces son stabil sürüm tercih edilecektir.
* Türksat tarafından belirtilen templateler projede kullanılabilir. (Ultima, Harmony vs…)
* Custom javascript Primefaces ihtiyaçlarına bağlı olarak yazılabilir fakat code-inline javascript yazılamaz. (Hariç: Localization En to Tr gibi…)
* Standart Html taglari yanlızca template yapısında kullanılmalı diğer sınıflarda mümkün olduğunca kullanılmamalıdır. Tüm tag ler JSF tag tree uzayında yer almalıdır.
* Mümkün oldukça jsf tag lib kullanılmalıdır, ihtiyaç halinde jstl kullanılabilir.
* Facelet kullanım ile sayfalar template mimarisine uygun tasarlanmalıdır. Merkezileştirilmiş template sayfa tasarımı ile ekranlar tasarlanmalıdır. Composition tasarım yaklaşımı ile sayfalar alt bloklara bölünerek, footer, content, head, menu gibi tüm sayfalarda ortak alanlar dikte edilmelidir.
* Bir ekran yada component birden fazla yerde kullanılacaksa composite komponent yapısıyla tasarlanmalıdır. Composite:interface, composite:attribute, composite:implementation unsurları dikkate alınarak tasarlanıp modülleştirilmelidir.

## Dış Kaynak Kullanım Kısıtları

Proje kapsamında kullanılacak tasarım öğeleri birtakım kısıtlara sahiptirler. Aşağıda gerekli kısıtlar listelenmiştir.

* Projelerde Bootstrap, Materialize CSS gibi frameworkler kullanılamaz. Bu gibi ihtiyaçlar öncelikle PrimeFaces kütüphanesi ile sağlanmalıdır.
* Tasarım öğeleri için uzak serverlardan kaynak çekilmemeli, lokal konumlandırma yapılmalıdır. Tüm modüller aşağıda belirtildiği gibi local adresten dosyaları çekeceklerdir.
  + Örn: <https://resources.[projeadi].gov>
* Özelleştirme için kullanılacak olan images, css ve js dosyaları resources klasörünün altında uygun klasör yapıları altına yerleştirilir.
  + Örn: /resources /rapor/css/style.css
  + Erişim : [https://resources.[projeadi].gov/resources /rapor/css/style.css](https://resources.[projeadi].gov/resources%20/rapor/css/style.css)
  + Örn: /resources /rapor/js/script.js
  + Erişim: [https://resources.[projeadi].gov/resources /rapor/js/script.js](https://resources.[projeadi].gov/resources%20/rapor/js/script.js)
  + Örn: /resources/rapor/images/image.png
  + Erişim: [https://resources.[projeadi].gov/resources /rapor/images/image.png](https://resources.[projeadi].gov/resources%20/rapor/images/image.png)
* GET Path parametreleri görünür olmamalıdır. İlgili parametreler hashlenip uygun şekilde gönderilmelidir.
* Url rewriting yapılarak real path gizlenmelidir. Bu işlem için PrettyFaces kullanılmalıdır.
* JSF form işlemleri ajax-enabled süreçte gerçekleştirmelidir. Ara yüzde bulunan tüm bileşenler back-end controller öğesi ile ajax üzerinden iletişime geçeceklerdir.

## Dikey Katman Tasarım Uygunlukları

AOP (Aspect Oriented Programming) tanım olarak yürüttükleri süreçleri ayrı olarak işleyen fakat kesişen davranışları birbirinden ayrıştırılması olarak tanımlanabilir. Projelerde modül bazlı tasarım ele alındığında benzer davranışları sürdüren iş parçalarını bir araya getiren alt birimler olarak adlandırılabilir. Bu aşamada odaklanılması gereken nokta “benzer davranış” olmalıdır. Bu benzer davranış ilgi alanlarının aynı olmasından veya ortak olmasından kaynaklanmaktadır. Veri yönetimi süreçlerinde bulunmayan bir sınıfın veritabanı işlemleri ile ilgisi olmayan bir method barındırması beklenemez. Bu başlık altında bulunan araçlar ilgili modül veya ilgili iş katmanının altında konumlandırılmalıdır. Dikey katman tasarımı (cross-cutting-concern) gerçekleştirilirken Spring AOP kütüphanesi kullanılacaktır. Bu süreçleri tanımlayacak ve tanımı genişletecek temel başlıklarla ele alalım.

### Yardımcı Unsurlar (Helpers)

Proje gelişim aşamasında tüm modüllerce ortak kullanım sınıflarına ihtiyaç bulunabilmektedir. Email gönderim, sms gönderim gibi ek kabiliyetleri destekleyici unsurları bu alanda topluyor, her modülün erişimine açıyoruz.

### Hata Yönetimi (Exception Handlers)

Sistemde gerekli kodlama süreçlerinde modüllere özgü hata tasarım ve indeksleme sürecine ihtiyaç bulunacaktır. Yazılım tasarımı esnasında her modül numaralandırıldığı için hata verdiği mesaj içeriği kapsamında modül numarasını ve ilgili problemi yansıtır içerikle mesaj vermelidir.

### Onaylayıcılar (Validations)

Ekran girdileri esnasında giriş denetleyicileri bu alanda bulur ve veri giriş kurallarını uygular. Tc, email, telefon, şifre vs giriş alanlarına uygun veri girişi yapıldığından emin olunmaktadır.

### Kayıt Altına Alma (Loging)

Her modül kendine ait bir numara ile oluşturulacaktır. Bu indekse uygun olarak hatanın sistematik bir şekilde numaralandırılması gerekmektedir. Kullanılacak olan log apisi Log4j yada Logback olarak sisteme işlenmelidir. Hataların mesaj formatı aşağıdaki gibi olmalıdır;

**Format**:

INFO/ERROR/DEBUG/WARNING-[ModulNo]-[zaman damgası]-[Hata Açıklaması]-[Tahmini çözüm indeks nosu]

SLF4J (Simple Logging Facade for Java) facade yapısına uygun log süreci yürütülmelidir. Bu sayede third-party log frameworkleri ile yazılım arasında soyutlama yaparak log frameworkleri arasında esnek geçiş imkanı vermektedir. Bütün log tanımlamaları SLF4J apisi kullanılarak yapılmalıdır Log4j yada Logback özel log tanımı yapılmamalıdır.

İsimlendirme üstlendiği görevi yansıtır şekilde olmalıdır ve numaralandırmaya M1 şekilde başlanıp devam edilmelidir (M1, M2 vs…). Verilmiş numaralar log çıktılarında kullanılacaktır.

**Örnek İsimlendirme**:

|  |  |  |
| --- | --- | --- |
| Modül Adı | Numara | Log Etiketi |
| Personel | M1 | M1-Personel |
| Rapor | M2 | M2-Rapor |

Şekil İsimlendirme ve loglama

### Ön Bellekleme

Projenin ihtiyaç duyduğu ön bellek destek araçları bu alanda konumlandırılır. Redis bu aşamada gerekli yardımcı araçlar ile modüllerin kullanımına sunulur. Redis Spring framework entegrasyonuyla RedisTemplate ile soyutlayarak kullanılmalıdır. Daha önce bahsedildiği gibi queryler ve sorgu sonuçlarını ehcache ile cachelenecektir. İhtiyaç duyulması durumunda session bilgisi, statik değerler, az değişen bilgiler Redis cache mekanizmasının son sürümü kullanılabilir.

# MVC Yaklaşımı ve Spring Framework

Uygulamada kullanılacak olan MVC framework’ü Spring Boot son stabil sürümüdür ve aşağıda bahsedilen standartlara uyularak kullanılmalıdır.

* Uygulama Apache Tomcat servlet containerında çalışabilmesi için uygulamada J2EE standartları kullanılmayacaktır. Bunun yerine Spring frameworkünün sunduğu J2EE özellikleri kullanılarak uygulama daha bağımsız ve hızlı çalışması sağlanmış olacaktır.
* Bir J2EE standardı olan JSF ManagedBean kullanılmayacaktır. Bunun yerine sunum katmanı için Spring bean annotasyonu olan @Controller kullanılacaktır.
* Uygulamada Spring beanleri doğru kullanımı sağlanacaktır. Generik tip olan @Component her yerde kullanılmamalıdır. Her katmanda katmana özgü sunulan Spring bean kullanılmalıdır ki hem kesin bir ayrıştırma sağlanmış olsun hem de spring frameworkünün bu beanler için oluşturduğu exception handling mekanizması düzgün çalışmış olsun. Bean kullanımı aşağıdaki tablodaki gibi olmalıdır;

|  |  |
| --- | --- |
| **Bean Tipi** | **Kullanılacağı Katman** |
| @Component | Jenerik tiptir. Uygulama genelinde kullanılacak bileşenler için uygundur. |
| @Controller | Sunum katmanı için kullanımı uygundur. |
| @Repository | DAO katmanı için kullanımı uygundur. |
| @Service | Servis katmanı için kullanımı uygundur. |

* Uygulamanın performansı ve güvenliği açısından Spring session scopelar(singleton,prototype,request v.b) doğru kullanımı sağlanmalıdır.
* Controller katmanında Springin sağladığı scoplarda bulunmayan view scope, spring scope’tan türetilerek oluşturulup kullanılmalıdır.
* Spring transaction yönetimi @Transactional notasyonu yardımıyla yalnızca business katmanında yönetilmelidir.
* Transaction yönetimi programatik değil dekleratif olmalıdır. Örneğin transaction.begin(), commit(), flush() gibi kodlanarak değil, @Transactional notasyonuyla birlikte propagation, isolation’ın doğru kullanımıyla yapılmalıdır.
* Spring AOP(Aspect Oriented Programming) modülüyle uygulamanın exception handling, logging gibi uygulamaya sıkıdan bağlı ve birbirleri ile kesişen kısımları daha modüler ve esnek yapılmalıdır.
* Uygulama geliştirirken ihtiyaç duyulan 3rd party teknoloji ihtiyaçları için Spring frameworkünün hazır sunduğu entegrasyonların (mongoDbTemplate, RedisTemplate, restTemplate v.b) kullanılması esas alınmıştır.

# Yazılım Geliştirme Teknolojileri Ve Uygunlukları

## Uygulanan Teknolojiler

Proje kapsamında değinilen teknolojiler aşağıdaki gibidir:

* Java 8 (Son Stabil Sürüm)
* Spring Boot (Son Stabil Sürüm)
* Hibernate (Son Stabil Sürüm)
* JSF (Mojarra Son Stabil Sürüm)
* PrimeFaces (Son Stabil Sürüm)
* Maven (Son Stabil Sürüm)
* Log4j / Logback (Son Stabil Sürüm)
* Mockito (Son Stabil Sürüm)
* JUnit (Son Stabil Sürüm)
* C3P0 / HIKARICP (Son Stabil Sürüm)
* EHCACHE (Son Stabil Sürüm)
* REDIS (Son Stabil Sürüm)
* ElasticSearch (Son Stabil Sürüm)
* MapStruct (Son Stabil Sürüm)
* PrettyFaces (Son Stabil Sürüm)
* SonarQube (Son Stabil Sürüm)
* Jasper Raports (Son Stabil Sürüm)
* JasperSoft Studio (Son Stabil Sürüm)
* Apache CXF (Son Stabil Sürüm)
* Swagger (Son Stabil Sürüm)
* Quartz Job Scheduler (Son Stabil Sürüm)
* Docker (Son Stabil Sürüm)
* Kubernetes (Son Stabil Sürüm)
* RabbitMQ (Son Stabil Sürüm)
* Camunda BPM(Son Stabil Sürüm)
* Camunda DMN(Son Stabil Sürüm)

## Modüler Tasarım Uygunlukları

Projenin modülleri iş katmanının alt parçalarıdır. Modüllerin bağımsız çalışabilme kabiliyetleri desteklenmelidir. Modüller amaçlarını yansıtır isimlendirmeler almalıdır. Her modül birbiri ile restful web servisler aracılığı ile iletişim kurmalıdır. Aşağıda modüllerin özellikleri belirtilmiştir. Maven modül yapısı baz alınarak tasarlanacak olan modüller gerekli bağımlılıkları yerleşik (local) Maven sunucusu üzerinden temin edeceklerdir.

Modül adlandırma örneği:

* personel
* personelrapor
* personelwebservis

### Modül Bazlı Servis Tasarımı

Yazılım katmanları arasındaki iletişim esnasında doğrudan entity objeleri kullanılmamalıdır. Bunun yerine DTO objeleri kullanarak daha az maliyetli nesne taşınmalıdır. Uygulama genelinde DTO ları entitylere mapleyen framework olarak MapStruct kullanılacaktır.

## Entegrasyon Testi

Entegrasyon testi JUnit ve SoapUI kullanılarak gerçekleştirilecektir. Araçların son stabil sürümleri kullanılacaktır.

## Birim Testi

Test süreçleri geliştirirken JUnit ve Mockito son stabil sürümleri kullanılacaktır.

## Log Araçları

Log işlemleri için kullanılacak apiler SLF4J ile birlikte log4j veya logback son stabil sürümleri kullanılacaktır .

## Statik Kod Analizi

Kodların analizi periyodik olarak SonarQube analiz aracı ile düzenli olarak standart Java ve FrontEnd kuralları ile kontrol edilecektir. İhtiyaca göre özelleşmiş kural setleri kullanılabilecektir.

## Raporlama

Proje kapsamında rapor üretimi için JasperReports, tasarım için Jaspersoft Studio kullanılacaktır. Raporlama süreçlerinde native sql kod kullanılmayacaktır. Raporun ihtiyaç duyduğu veri, servis katmanında hazırlanarak rapora geçilecektir. Böylelikle süreç içinde veritabanı bağımlılığı minimize edilecektir. Raporlama süreçlerinden sorumlu ayrı bir modül oluşturulacaktır. Bu modül ile etkileşim halinde olan tüm alt modüller servisler aracılığı ile gerekli parametreleri iletiyor olacaklardır. Üretilmiş tüm raporlar sistem tarafından log altına alınacaktır. Log içeriğinde kimin, ne zaman, kaç numaralı modül ve hangi başlıklı rapor alındığının bilgisi bulunacaktır.

Log formatı: [kullanıcı\_id]-[zaman damgası]-[modül no (M1, M2 gibi…)]-[rapor başlığı]

Örn: 111-20.05.2018 16:21:00-M1-Örnek Kayıt Listesi

## Servis Katmanı Ve Tasarımı

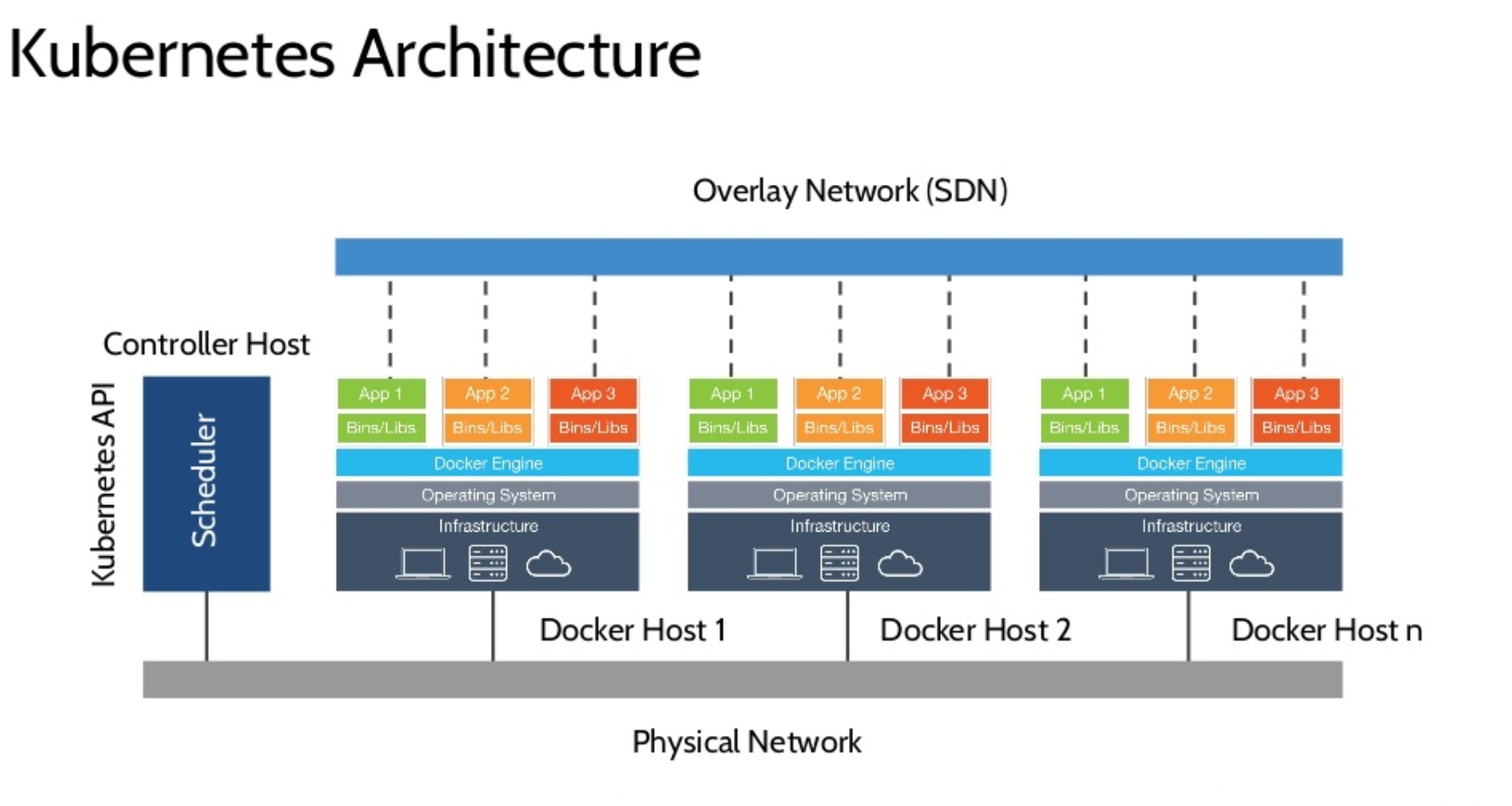
Servis mimarisi kapsamında kullanılacak ve üretilecek SOAP servisleri için Apache CXF framework yapısı kullanılacaktır. Notasyonel kullanım belirteçleri JAX-WS standartları üzerinden gidilmelidir. Restful servis kapsamında Spring Rest mimarisi kullanılarak gerekli servisler üretilecektir ve restful web servislerinin dokümantasyonu için Swagger kullanılacaktır.

## Zamanlanmış Görev Kapsamı

Zamanlanmış görevler süreç içinde sistemin birtakım kontrol mekanizmalarını kullanan veya istenilen görevleri düzenli gerçekleştiren işlemler bütünüdür. Bu bütünleşik sistem mekanizmasında kullanılacak olan zamanlanmış görev framework için Spring Quartz entegrasyonu tercih edilecektir.

## Konteyner Uygunlukları

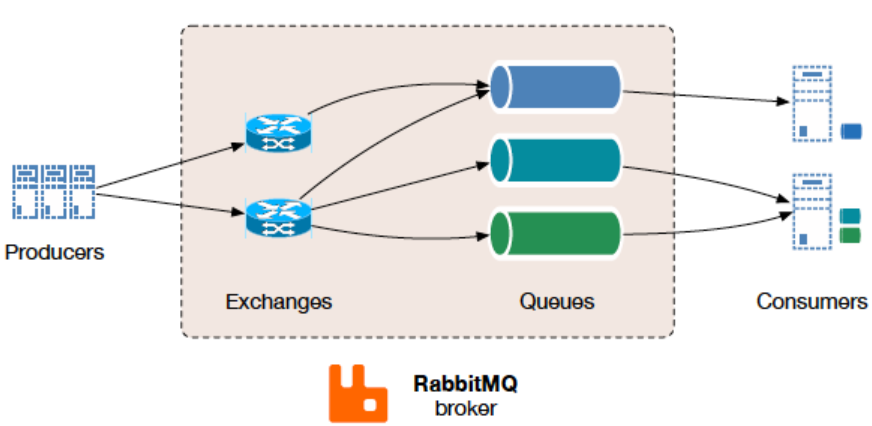
Yazılım geliştirme sürecinde uygulamanın derlenmesi, taşınması ve çalıştırılması işini yerine getirmeyi sağlayan Docker “Package Once Deploy Anywhere” (PODA) paradigmasını benimsediği için tercih edilecektir. Docker’ın scaling, cluster ve load balance gibi birtakım araçlara ihtiyaç duyulduğu süreçte Docker Kubernetes entegrasyonu kullanılmalıdır.



Şekil Docker konteyner yapısı

## Mesaj Kuyruğu Sistemi

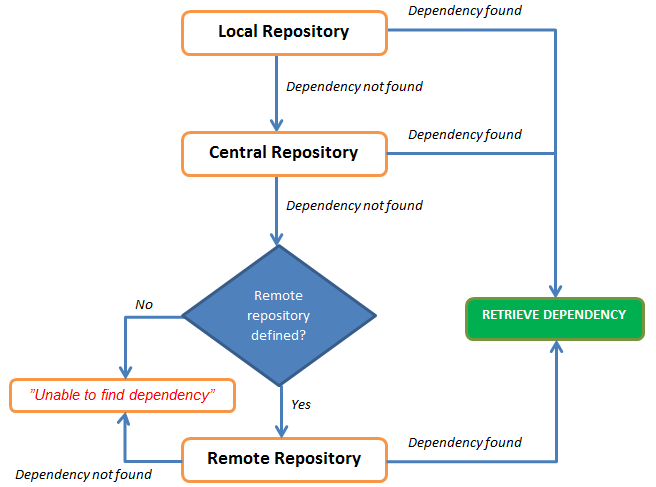
Sistemin herhangi bir kaynaktan aldığı bir mesajın, bir başka kaynağa sırası geldiğinde iletilmesi süreçlerini yönetmektedir. Mesaj kuyruğu sistemi olarak RabbitMQ tercih edilecektir. RabbitMQ Spring entegrasyonu ile birlikte RabbitTemplate olarak sunulan soyutlanmış sınıf aracılığıyla kullanılmalıdır.



Şekil Basitçe çalışma mimarisi

## Bağımlılıkların Yönetimi

Projede ihtiyaç duyulan kütüphaneler Maven ile yönetilecektir. Dış kaynak maven repository lerine erişim yapılmadan, local repository kullanımı ile bağımlılıklar kullanılacaktır. Bu süreci yönetmek için Maven server kurulumu yapılarak ilgili bağımlılıklar kurulu server üzerinden çekilecektir.



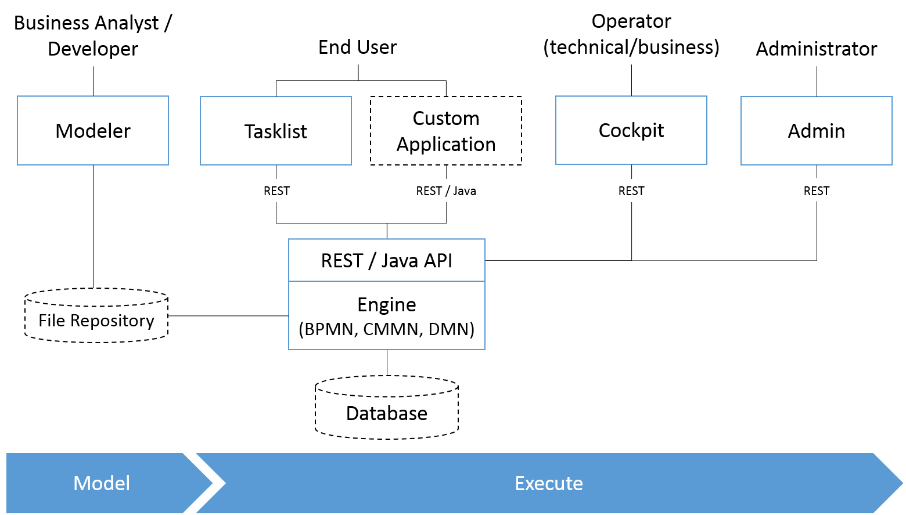
Şekil Maven sunucusunun çalışma şeması

## Dinamik İş Süreçleri Ve Kuralları

### BPM

Kurumsal Süreç Yönetimi (Business Process Management) şeklinde Türkçeye çevirebileceğimiz bu kavram işletmelerin işlerini yapma biçimlerini modellemek için kurum içinde yapılan işleri bütünsel bir yaklaşımla ele alan bir yöntemdir. BPM işletmelerin varlıkları arasında sayılabilecek süreçlerin ortaya çıkarılması, belgelendirilmesi, iyileştirmelerin yapılmasını ve adımların otomatik yapılmasını hedefler. Bu hedefe odaklanmış ve kullanıma sunulmuş birçok açık kaynak süreç yönetim aracı bulunmaktadır. Bu araçlar arasında bulunan, dinamik süreçleri yönetmek ve modellemek için Camunda kullanılacaktır. Camunda’nın özellikleri kısaca şöyledir;

* **Çekirdek Motor:** Camunda BPM’nin çekirdeği, süreç otomasyonu için OMG standartları BPMN 2.0, vaka yönetimi için CMMN 1.1 ve karar yönetimi için DMN 1.1 destekleyen bir model yürütme altyapısıdır.
* **Uygulamalar:** Camunda BPM, Camunda çekirdek motorunda çalışan süreç uygulamalarını modellemenize, yürütmenize ve yönetmenize yardımcı olacak bir dizi uygulama ile birlikte gelir. Bu uygulamalar çekirdek motorunun genel REST API’sı ile etkileşime girer. Son stabil sürümde, çekirdeğin motorunun genel Java API’sini kullanarak kendi uygulamalarınızı da oluşturmaya imkan tanıyor fakat BPM süreçlerinde java apilerini yoğun olarak kullanımı ve **business katmanındaki mantığın BPM üzerine taşınması kabul edilemez bir tercihtir**. BPM sadece süreç otomasyonuna odaklanmalıdır.

****

Şekil 9Camunda mimarisi ile ilgili bir görsel.