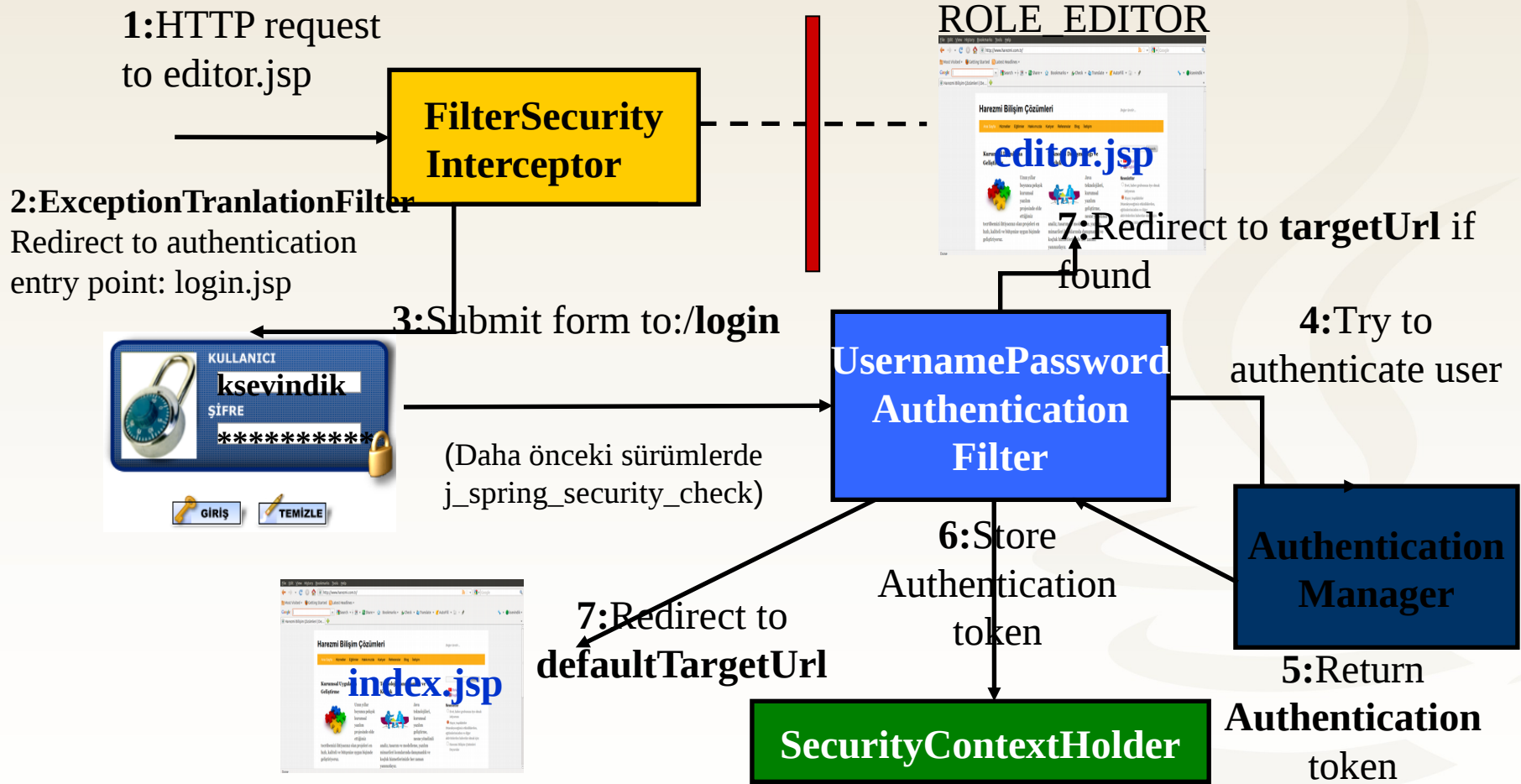
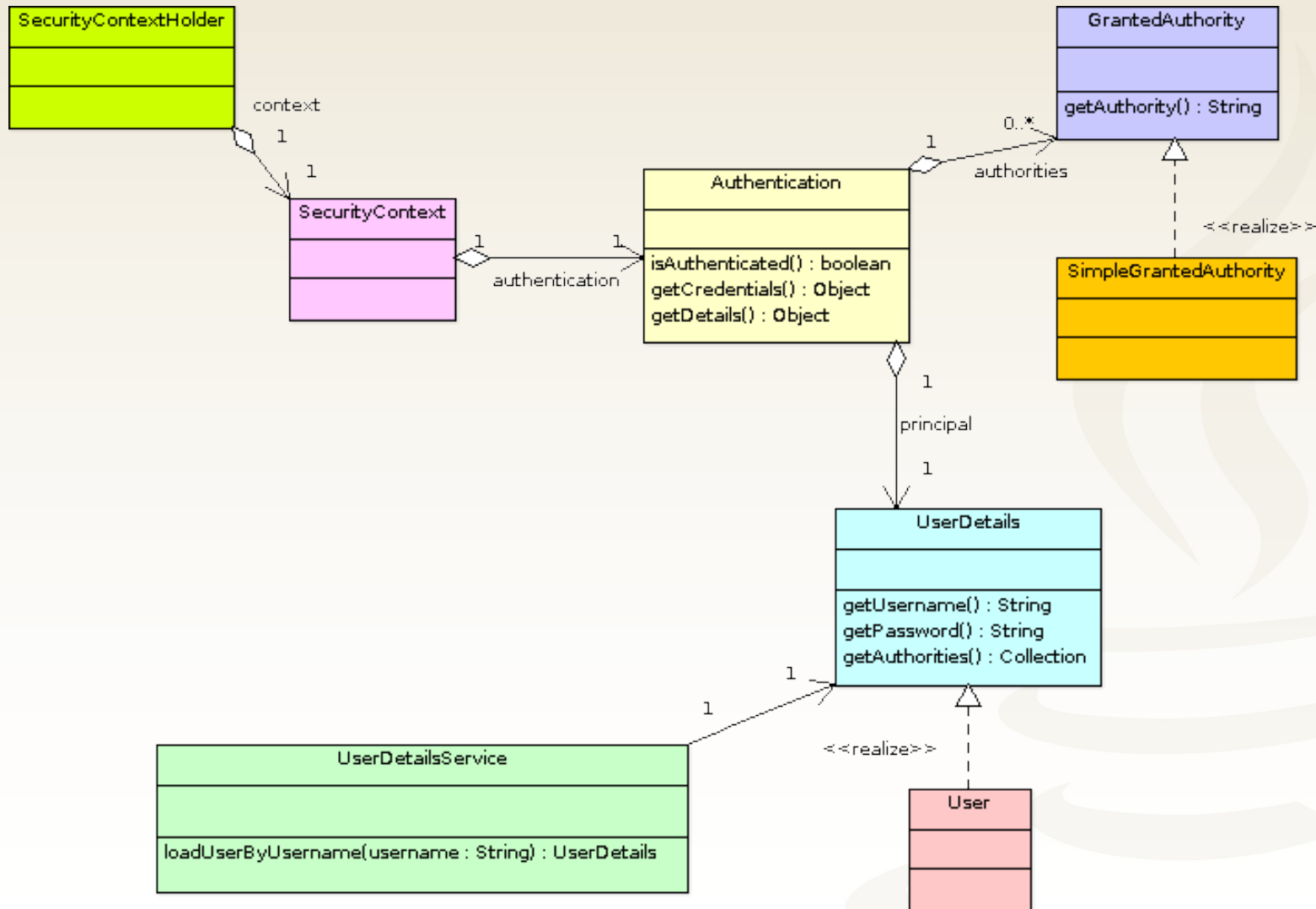


İleri Düzey Spring Eğitimi Security

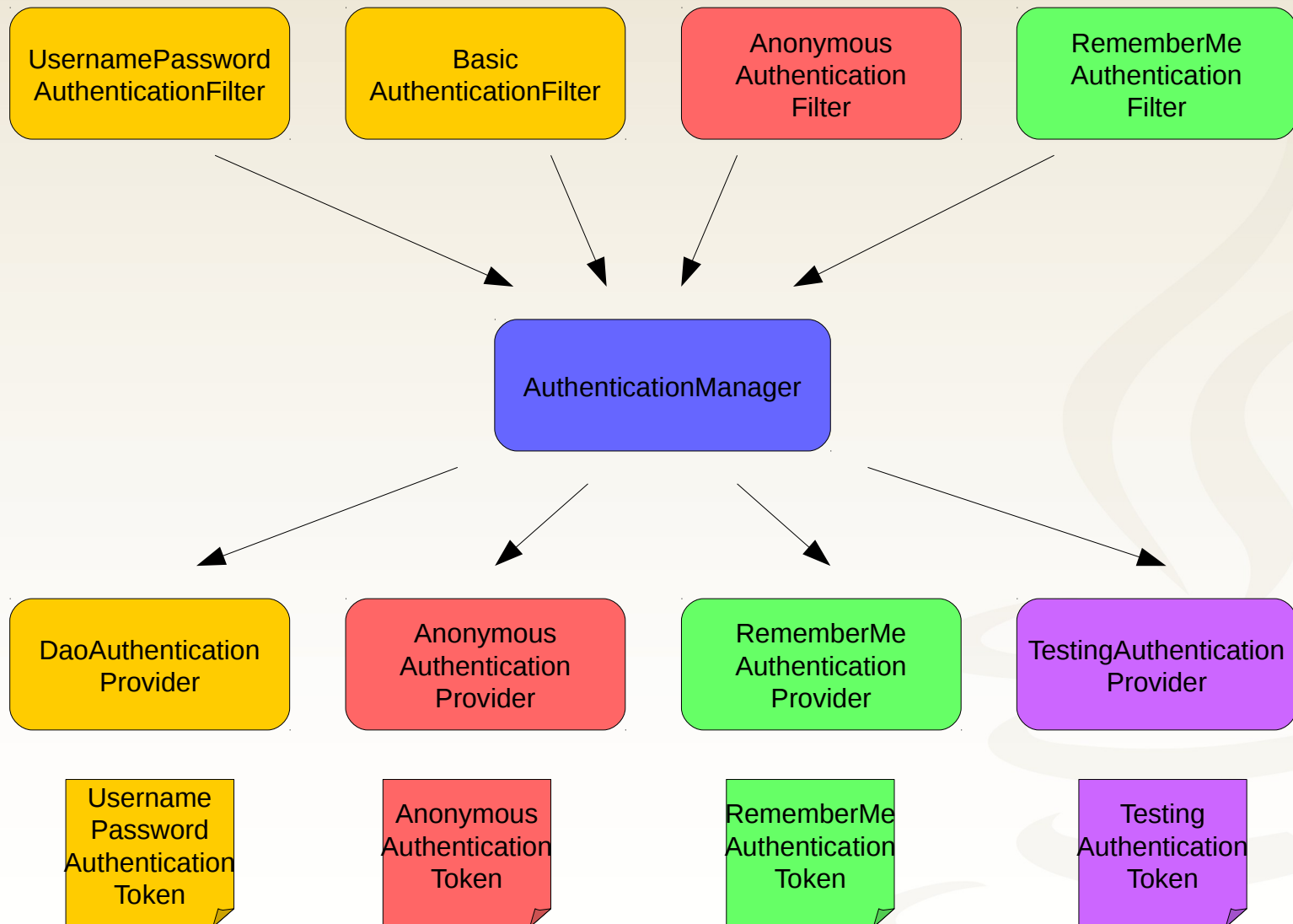
Kimliklendirmenin İşleyişi



Spring Security Domain Sınıfları



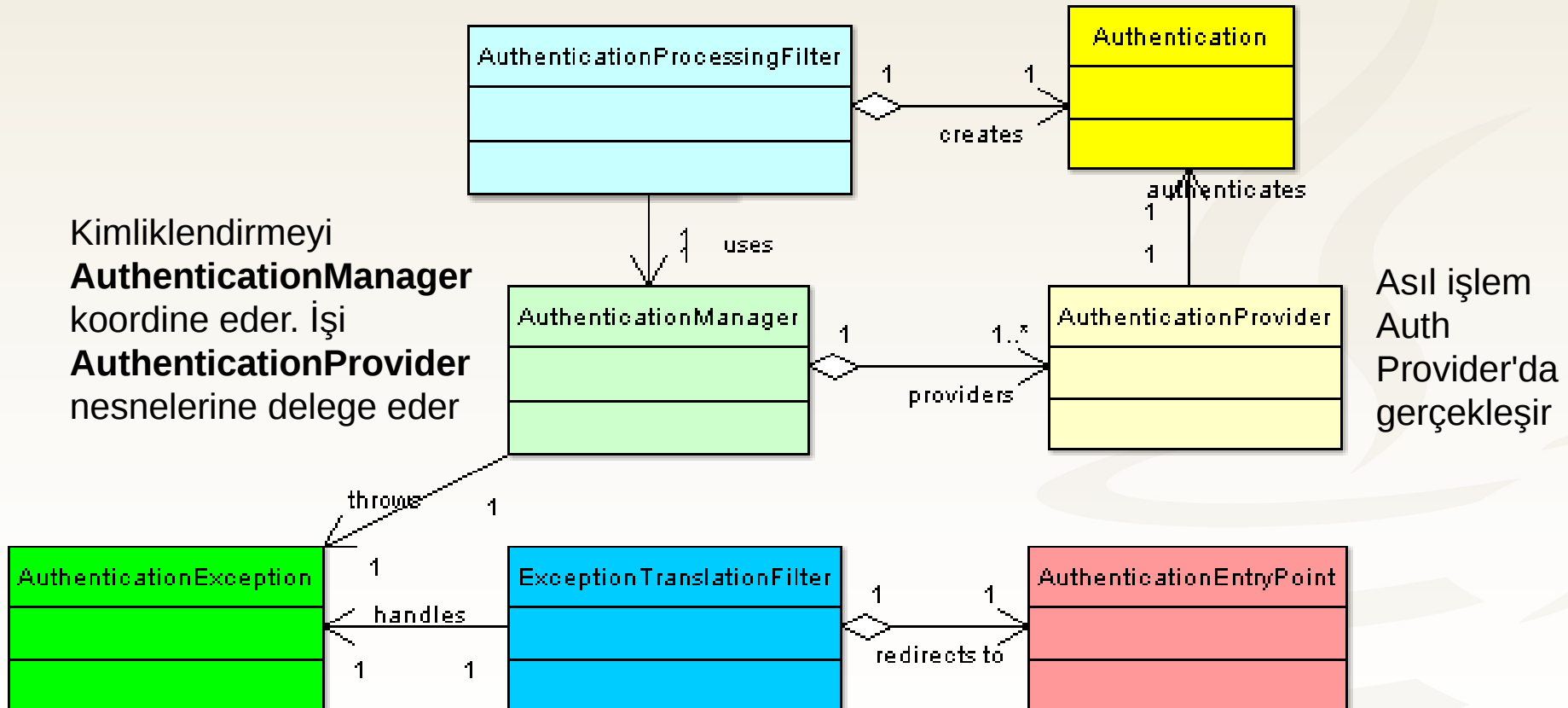
Kimliklendirme Modeli



Kimliklendirme Modeli

Kimliklendirmenin başladığı yer **AbstractAuthenticationProcessingFilter**'dir. Her kimliklendirme yönteminin kendine özel bir sub class'ı mevcuttur

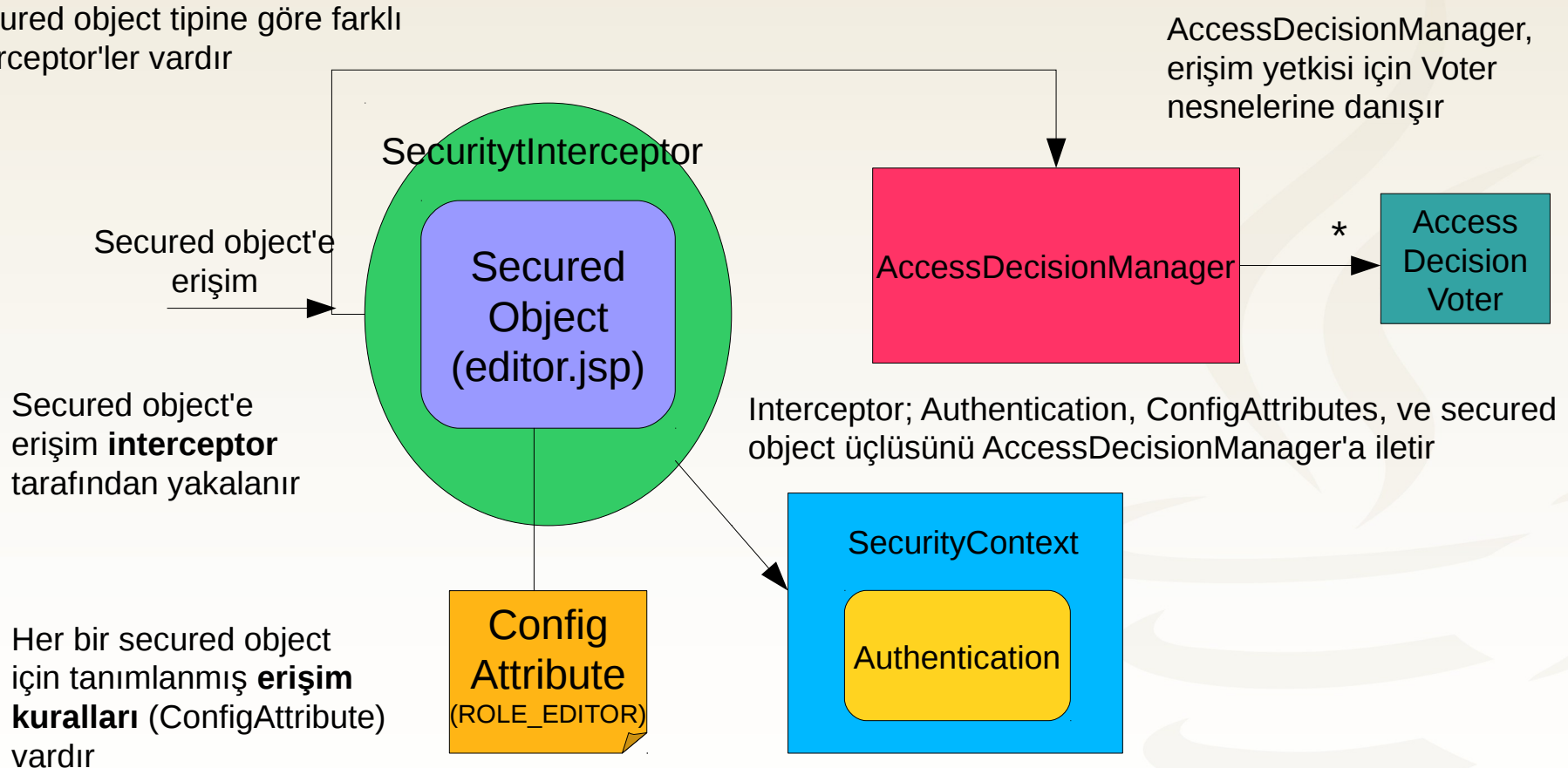
Başarılı kimliklendirme sonucu **Authentication** nesnesi oluşur
Farklı auth yöntemlerinin kendine özel token impl mevcuttur



Her auth yönteminin kendine özel bir impl mevcuttur

Yetkilendirmenin İşleyişi

Secured object tipine göre farklı interceptor'ler vardır



Container'da Birden Fazla <security:http> Elemanı

- Spring konfigürasyon dosyasında aynı anda birden fazla `<security:http>` elemanı tanımlanabilir
- Bunun amacı uygulama içerisinde **farklı URI pattern'larını farklı kimliklendirme metotları** ile ele almaktır
- Ya da her bir URI pattern'ında **farklı güvenlik kontrollerini** devreye sokmaktır

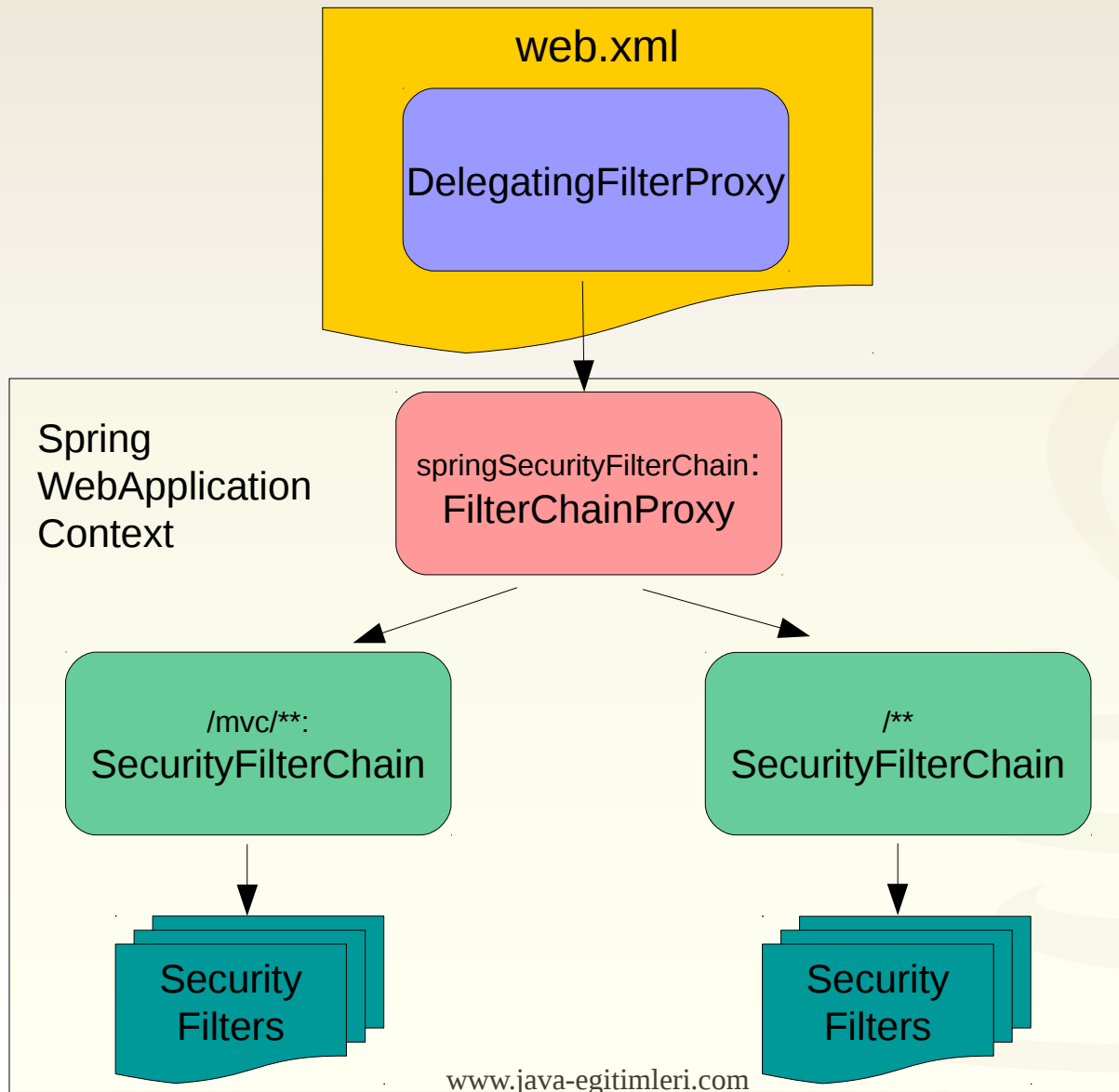
Container'da Birden Fazla <security:http> Elemanı

```
<beans...>
  <security:http pattern="/mvc/**" create-session="stateless">
    <security:http-basic />
    <security:intercept-url pattern="/**" access="hasRole('ROLE_USER')"/>
  </security:http>

  <security:http pattern="/**" auto-config="true">
    <security:access-denied-handler
      error-page="/accessDenied.jsp" />

    <security:form-login login-page="/login.jsp"
      authentication-failure-url="/login.jsp?error=true" />
    <security:intercept-url pattern="/login.jsp"
      access="isAnonymous()" />
    <security:intercept-url pattern="/editor.jsp"
      access="hasRole('ROLE_EDITOR')" />
    <security:intercept-url pattern="/**"
      access="hasRole('ROLE_USER')" />
  </security:http>
</beans>
```


Spring Security Filter Mimarisi



HttpServletRequest API'nin Kullanılması

- Mevcut kullanıcının kimlik bilgisine erişmek, yetki kontrolü yapmak için

HttpServletRequest bir takım metotlar sunmaktadır

- `getRemoteUser():String`
 - `getUserPrincipal():Principal`
 - `isUserInRole(roleName:String):boolean`
- Spring Security, ilgili bir filter ile current **request nesnesini wrap ederek** bu metotların uygun cevaplar dönmesini sağlamaktadır

HttpServletRequest

Nesnesine Erişim

- Spring Framework ise current request nesnesini **ThreadLocal** bir değişkende tutarak **RequestContextHolder** ile uygulama içerisinde **herhangi bir katmandan erişilmesini** sağlar
- Current request'i ThreadLocal değişkene **bind etmek** için Spring üç farklı yöntem sunar
 - DispatcherServlet
 - RequestContextListener
 - RequestContextFilter

HttpServletRequest Nesnesine Erişim

- **DispatcherServlet** üzerinden erişilen request'lerde (Spring MVC REST vb) herhangi bir sorun yoktur
- Ancak **RequestContextListener**, current request Spring Security Filter Chain'e daha girmeden evvel ThreadLocal'e bind edilmektedir
- Request bu aşamada daha **SecurityContextHolderAware RequestFilter** tarafından wrap edilmemiş vaziyettedir

HttpServletRequest Nesnesine Erişim

- Dolayısı ile request API üzerinden erişilen güvenlik metotları **istenen biçimde çalışmayacaktır**
- Yapılması gereken **web.xml'de RequestContextFilter'ı** tanımlayıp Spring Security Filter Chain'den **sonra** devreye girmesini sağlamaktır
- Böylece current request nesnesi **wrap edilmiş olacak**, `isUserInRole()` gibi metotlarda **Spring Security altyapısından cevap** dönecektir

RequestContextFilter Konfigürasyonu

```
<web-app...>
  ...
  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
      org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
  </filter>
  <filter>
    <filter-name>requestContextFilter</filter-name>
    <filter-class>
      org.springframework.web.filter.RequestContextFilter
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/app/*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>requestContextFilter</filter-name>
    <url-pattern>/app/*</url-pattern>
  </filter-mapping>
</web-app>
```

Spring Security Özelleştirmeleri

- Spring Security oldukça **esnek** ve uygulamalara göre **özelleşmeye açık** bir framework'tür
- Kimliklendirme ve yetkilendirme **süreçlerinin hemen her adımı** uygulamaya özgü gereksinimlere göre özelleştirilebilir
- Custom login form, custom AuthenticationProvider, Custom UserDetailsService ve UserDetails sınıfları, Filter zincirinde değişiklik **en yaygın özelleştirme örnekleridir**

Custom Login Form


- Örneğin, login formunda kullanıcı adı, şifrenin yanı sıra birde parola sorulabilir
- Ya da captcha özelliği eklenmesi istenebilir
- Bu gibi ihtiyaçlar genellikle **UsernamePasswordAuthenticationFilter** sınıfından extend ederek karşılanabilir
- **UsernamePasswordAuthenticationFilter.attemptAuthentication()** metodu **override** edilerek özelleştirme yapılabilir

Custom Login Form

```
<beans>
  <bean id="customizedFormLoginFilter"
        class="x.y.z.CustomFormLoginFilter">
    <property name="authenticationManager"
              ref="authenticationManager"/>
    ...
  </bean>

  <security:authentication-manager alias="authenticationManager">
    ...
  </security:authentication-manager>

  <security:http>
    <security:custom-filter ref="customizedFormLoginFilter"
                          position="FORM_LOGIN_FILTER"/>
  </security:http>
</beans>
```



AuthProvider

- Uygulamaya özel, örneğin, kullanıcının eriştiği istemci makinanın belirli bir IP grubunda olması gibi Authentication kontrolleri eklenebilir
- Bu gibi durumlarda **DaoAuthProvider** sınıfını extend etmek, ya da **AuthProvider** arayüzünü implement etmek gerekebilir
- **DaoAuthProvider.additionalAuthenticationChecks()** metodu override edilerek bu kontroller yapılabilir


Custom AuthenticationProvider

```
<beans>
  <security:authentication-manager>

    <security:authentication-provider
      ref="customAuthProvider"/>

  </security:authentication-manager>

  <bean id="customAuthProvider"
    class="x.y.z.CustomAuthenticationProvider">
    ...
  </bean>
</beans>
```



Filter Zincirine Ekleme

- Örneğin, belirli durumlar için kimliklendirmenin bypass edilmesi istenebilir
- Ya da kullanıcı authenticated iken login sayfasına eriştiğinde ana sayfaya yönlendirilmesi istenebilir
- Yeni bir **Filter yazılması** ve **springSecurityFilterChain'e** eklenmesi gerekebilir

Filter Zincirine Ekleme

```

<beans>
  <bean id="authByPassFilter" class="x.y.z.AuthByPassFilter">
    ...
  </bean>

  <bean id="mainPageRedirectFilter"
        class="x.y.z.MainPageRedirectFilter">
    ...
  </bean>

  <security:http auto-config="true">
    ...
    <security:custom-filter ref="authByPassFilter"
                           before="FORM_LOGIN_FILTER"/>

    <security:custom-filter ref="mainPageRedirectFilter"
                           after="SECURITY_CONTEXT_FILTER"/>

  </security:http>
</beans>

```

Custom UserDetails ve UserDetailsService

- Örneğin, uygulama kendine ait bir User tipini kullanmak isteyebilir
- Ya da Authentication bilgisinin bir kısmı DB'den diğer bir kısmı LDAP veya başka bir realm'den gelebilir
- Bu gibi durumlarda **UserDetails** ve **UserDetailsService** arayüzlerinin implement edilmesi gerekecektir

Custom UserDetails ve UserDetailsService

```
<beans>
  <security:authentication-manager>

    <security:authentication-provider
      user-service-ref="customUserDetailsService">

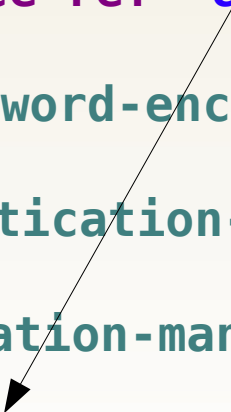
      <security:password-encoder hash="sha" />

    </security:authentication-provider>

  </security:authentication-manager>

  <bean id="customUserDetailsService"
    class="x.y.z.CustomUserDetailsService">

  ...
</bean>
</beans>
```



FilterChainProxy'nin Explicit Konfigürasyonu

- Spring Container içerisinde en az bir http elemanı tanımlanması
springSecurityFilterChain isminde ve **FilterChainProxy** tipinde bir bean yaratılmasını tetikler
- Birden fazla http elemanının her biri de bu bean altında ayrı ayrı **SecurityFilterChain** bean'leri şeklinde ele alınmaktadır

FilterChainProxy'nin Explicit Konfigürasyonu

- **FilterChainProxy** bean'i, hiç **security:http** elemanı olmadan normal **bir bean** şeklinde de tanımlanabilir

```
<bean id="filterChainProxy" class="org.springframework.security.web.FilterChainProxy">
  <constructor-arg>
    <list>
      <security:filter-chain pattern="/mvc/**"
        filters="securityContextPersistenceFilterWithASCFALSE,
        basicAuthenticationFilter,exceptionTranslationFilter,
        filterSecurityInterceptor" />
      <security:filter-chain pattern="/**"
        filters="securityContextPersistenceFilterWithASCTrue,
        formLoginFilter,exceptionTranslationFilter,
        filterSecurityInterceptor" />
    </list>
  </constructor-arg>
</bean>
```

Bu durumda web.xml'deki DelegatingFilterProxy filter tanımının ismi de bean ismi ile eşleşmelidir.

SecurityMetadataSource'un Özelleştirilmesi

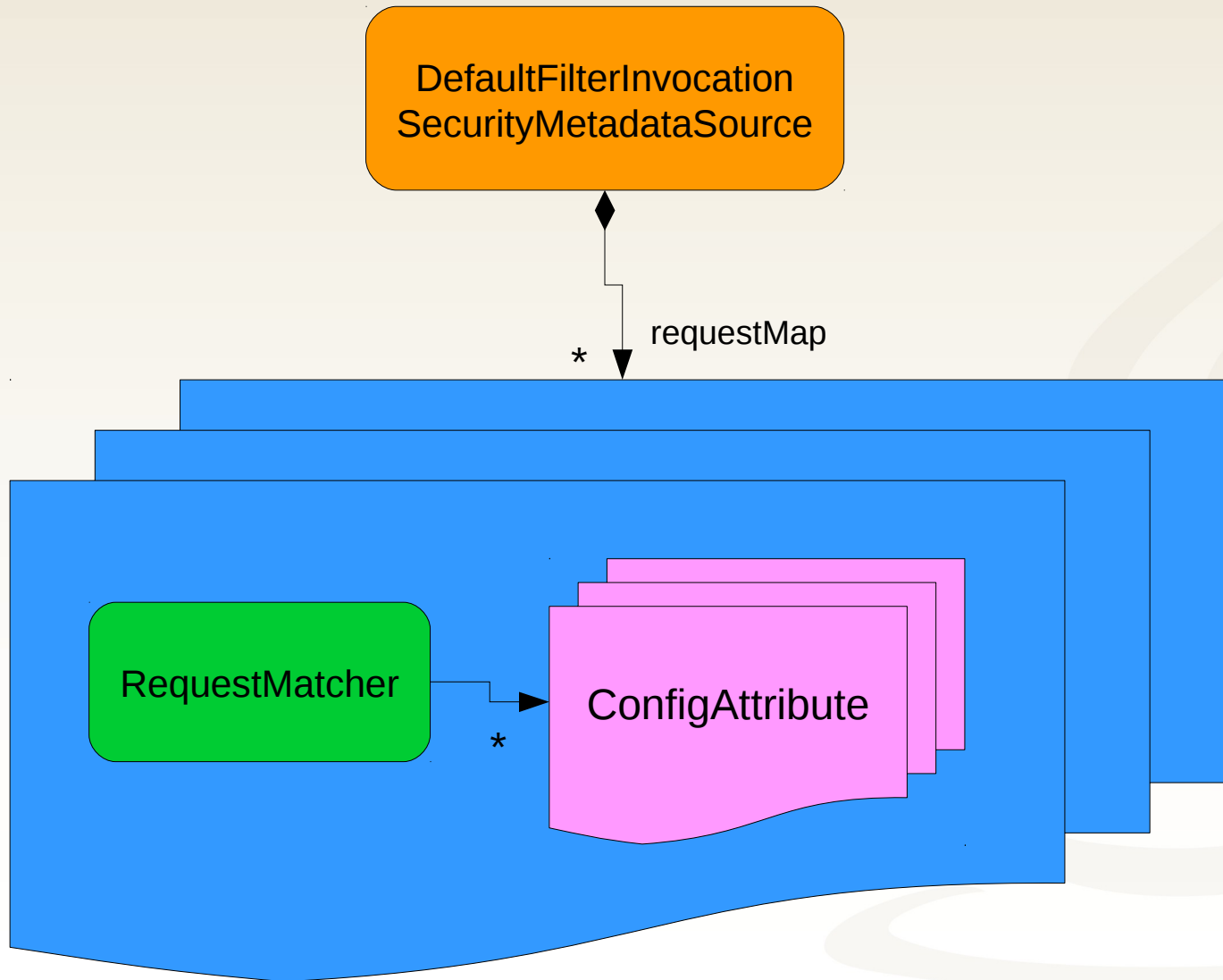
- **security:http** elemanı kullanılmayan durumlarda genellikle **FilterSecurityInterceptor** bean'ine enjekte edilen **SecurityMetadataSource'un** da özelleşmesi söz konusu olmaktadır

```
<bean id="filterSecurityInterceptor"  
  class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor">  
  
  <property name="authenticationManager" ref="authenticationManager" />  
  <property name="accessDecisionManager" ref="accessDecisionManager" />  
  <property name="securityMetadataSource">  
    <security:filter-security-metadata-source use-expressions="false">  
      <security:intercept-url pattern="/**" access="IS_AUTHENTICATED_FULLY" />  
    </security:filter-security-metadata-source>  
  </property>  
</bean>
```

SecurityMetadataSource'un Özelleştirilmesi

- Ya da **intercept-url** tanımlarının **veritabanı** gibi bir yerde yönetilmesi söz konusu olabilir
- Bu durumda **DefaultFilterInvocationSecurityMetadataSource** sınıfından bir bean oluşturularak içeriği DB'den elde edilmelidir
- Daha sonra bu bean **FilterSecurityInterceptor** bean'ine enjekte edilmelidir

SecurityMetadataSource'un DB Entegrasyonu



SecurityMetadataSource İçin Örnek Veri Modeli

REQUEST_MATCHERS_TABLE

ID	ORDER	REQUEST_MATCHER_PATTERN
1	0	/vets.jsp
2	1	/createVet.jsp
3	2	/**

CONFIG_ATTRIBUTES_TABLE

ID	PATTERN_ID	CONFIG_ATTR_VALUE
1	1	ROLE_USER
2	1	ROLE_EDITOR
3	2	ROLE_EDITOR
4	3	IS_AUTENTICATED_FULLY

RequestMatcher ve ConfigAttribute Arayüzleri

- **RequestMatcher** arayüzü için
AntPathRequestMatcher,
RegexRequestMatcher gibi sınıflar
kullanılabilir
- **ConfigAttribute** arayüzü için
SecurityConfig veya
WebExpressionConfigAttribute sınıfları
kullanılabilir

RequestMatcher Örnekleri

- `/**` gibi bir RequestMatcher değeri **AntPathRequestMatcher** ile ifade edilir
- `/find.*` gibi bir değer ise **RegexRequestMatcher** ile ifade edilir
- Herhangi bir request URI ile eşleşmesi için **AnyRequestMatcher** kullanılabilir

ConfigAttribute Örnekleri

- `ROLE_USER`, `IS_AUTHENTICATED_FULLY` gibi değerler **SecurityConfigAttribute** ile ifade edilirler
- `hasRole('ROLE_USER')`, `permitAll`, `isFullyAuthenticated()` gibi SpEL değerleri ise **WebExpressionConfigAttribute** ile ifade edilirler
- **SpelExpressionParser** ile `WebExpressionConfigAttribute`'un beklediği `Expression` nesne oluşturulabilir

İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com

