

# Spring Boot Framework

## 3



# Spring Boot ve Güvenlik

- Spring Security kabiliyetlerini devreye almak için öncelikle **security starter**'i tanımlı olmalıdır

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

- Default olarak **bütün HTTP endpoint'leri** güvenlik altına alınmıştır

# Spring Boot ve Güvenlik

- Default olarak **password'ü rastgele belirlenmiş** ve INFO log düzeyinde console'a yazılan **user** isimli bir kullanıcı mevcuttur
- Bu kullanıcı adını ve password'ü ve rollerini application.properties'den değiştirmek mümkündür

```
security.user.name=user  
security.user.password=secret  
security.user.role=USER,ACTUATOR
```

# Spring Boot ve Güvenlik

- HSTS, XSS, CSRF gibi özellikler devreye alınmıştır
- Default olarak aşağıdaki statik web resource path'leri **public resource** (unsecure) olarak erişilmektedir
  - /css/\*\*
  - /js/\*\*
  - /images/\*\*
  - /webjars/\*\*
  - \*\*/favicon.ico

# Default Security Konfigürasyonu

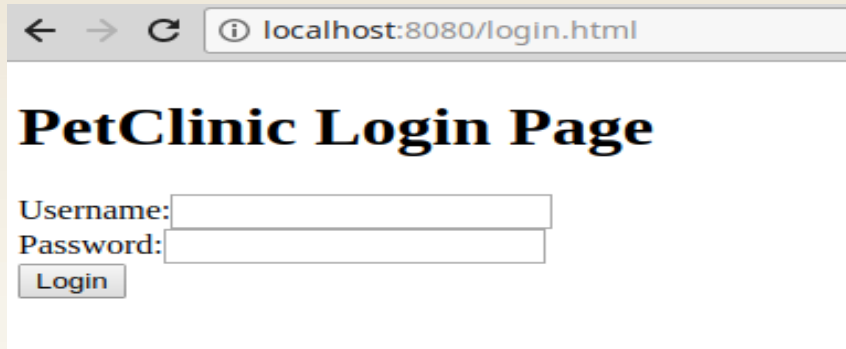
- Default web app security konfigürasyonunu özelleştirmek için **WebSecurityConfigurerAdapter** sınıfından türeyen bir konfigürasyon sınıfı yazılmalıdır
- Default konfigürasyonu tamamen devre dışı bırakmak için **@EnableWebSecurity** eklenmelidir

@Configuration

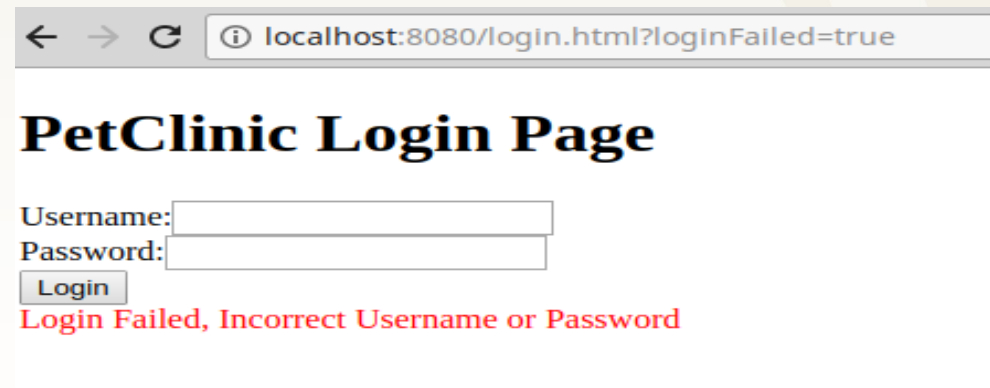
@EnableWebSecurity

```
public class PetClinicSecurityConfig extends WebSecurityConfigurerAdapter {  
    ...  
}
```

# Form Tabanlı Login - Overview



A screenshot of a web browser window showing the 'PetClinic Login Page'. The address bar displays 'localhost:8080/login.html'. The page has a title 'PetClinic Login Page' in a large, bold, black serif font. Below the title, there are two input fields: 'Username:' and 'Password:'. The 'Username:' field is a single-line text box, and the 'Password:' field is a single-line text box. Below the 'Password:' field is a 'Login' button with a grey background and black text.



A screenshot of a web browser window showing the 'PetClinic Login Page' after a failed login attempt. The address bar displays 'localhost:8080/login.html?loginFailed=true'. The page has a title 'PetClinic Login Page' in a large, bold, black serif font. Below the title, there are two input fields: 'Username:' and 'Password:'. The 'Username:' field is a single-line text box, and the 'Password:' field is a single-line text box. Below the 'Password:' field is a 'Login' button with a grey background and black text. Below the 'Login' button, there is a red error message: 'Login Failed, Incorrect Username or Password'.

# Form Tabanlı Login - Controller

http://localhost:8080/login.html



```
@Controller  
public class LoginFormController {
```

```
@RequestMapping(value="/login.html",method=RequestMethod.GET)  
public ModelAndView loginPage() {
```

```
ModelAndView mav = new ModelAndView();  
mav.setViewName("login");  
return mav;
```

```
}
```

```
}
```

application.properties

```
spring.mvc.view.prefix=/WEB-INF/jsp/  
spring.mvc.view.suffix=.jsp
```

/WEB-INF/jsp/login.jsp

# Form Tabanlı Login - JSP

/WEB-INF/jsp/login.jsp



```
<html>
<body>
    ...
    <form action="login" method="post">

        Username:<input name="username" type="text" /> <br/>

        Password:<input name="password" type="password" /> <br/>

        <input type="hidden"
                name="${_csrf.parameterName}"
                value="${_csrf.token}"/>

        <input type="submit" value="Login"/>
    </form>
</body>
</html>
```



# Form Tabanlı Login – Config

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
{
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/**/favicon.ico",
                "/css/**", "js/**", "/img/**",
                "/webjars/**", "/login.html").permitAll()

        .anyRequest().authenticated();

        http.formLogin()
            .loginPage("/login.html")
            .loginProcessingUrl("/login")
            .failureUrl("/login.html?loginFailed=true");
    }
}
```

# Logout Kabiliyeti

- Spring Security'nin default ayarlarında **logout ve CSRF kabiliyetleri aktiftir**
- Default logout URI'ı **/logout** şeklindedir
- Ancak **CSRF**, /logout URI'ına **HTTP GET** ile erişime **izin vermez**
- Dolayısı ile **logout işlemi** için bir **HTML form** kullanmak ve **POST** metodu ile web isteği göndermek gerekecektir

# Logout Kabiliyeti - JSP

```
<html>
  <body>
    ...
    <form action="logout" method="post">

      <input type="submit" value="Logout">

      <input type="hidden"
              name="${_csrf.parameterName}"
              value="${_csrf.token}">

    </form>
  </body>
</html>
```

# Beni Hatırla Kabiliyeti - Overview

**PetClinic Login Page**

Username:

Password:

Remember Me: ☒

Login

# Beni Hatırla Kabiliyeti - JSP

```
<html>
  <body>
    ...
    <form action="login" method="post">

      Username:<input name="username" type="text" /> <br/>

      Password:<input name="password" type="password" /> <br/>

      Remember Me:<input name="remember-me" type="checkbox"> <br/>

      <input type="hidden"
        name="${_csrf.parameterName}"
        value="${_csrf.token}" />

      <input type="submit" value="Login"/>
    </form>
  </body>
</html>
```

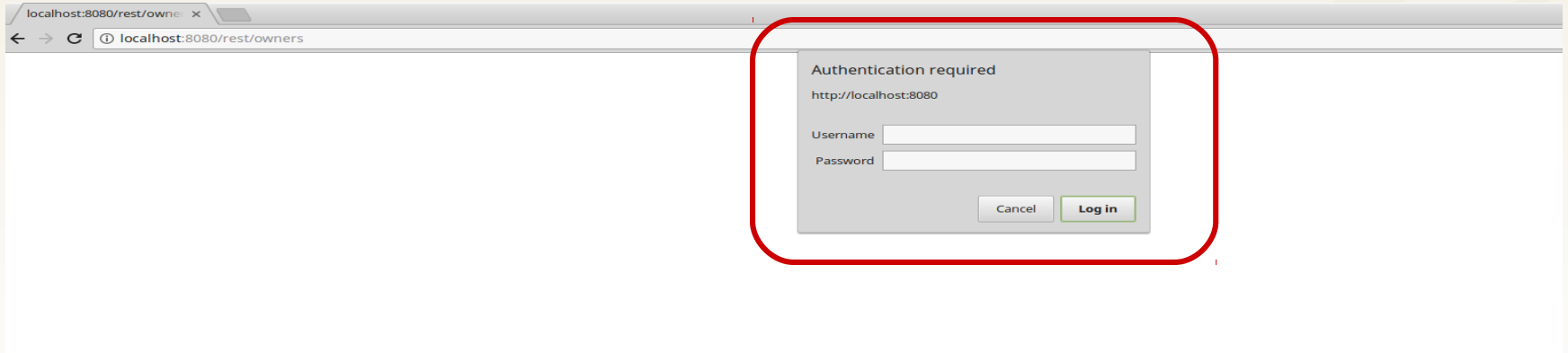
# Beni Hatırla Kabiliyeti - Config

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
{

    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        ...
        http.rememberMe().userDetailsService(userDetailsService);
    }
}
```

# HTTP Basic Auth - Overview



# HTTP Basic Auth - Config

```
@Configuration
public class SecurityConfiguration extends
    WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        ...
        http.httpBasic();
    }
}
```



# H2 Console'a Erişim

- H2 Console'un kendine ait bir authentication mekanizması olduğu için **spring security'nin dışında bırakılması** gerekir

```
@Configuration
@Order(value=0)
public class H2SecurityConfiguration
    extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.antMatcher("/h2-console/**");
        http.authorizeRequests().anyRequest().permitAll();
        http.csrf().disable();
        http.headers().frameOptions().disable();
    }
}
```

# Web Kaynaklarının Yetkilendirilmesi

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests()

            .antMatchers(
                "**/favicon.ico", "/css/**", "js/**", "/images/**",
                "/webjars/**", "/login.html").permitAll()

            .antMatchers("/rest/**").access("hasRole('EDITOR')")

            .antMatchers("/actuator/**").access("hasRole('ADMIN')")

            .anyRequest().authenticated();
    }
}
```



# Metot Düzeyinde Yetkilendirme

- Metot düzeyinde yetkilendirmeyi devreye almak için

**@EnableGlobalMethodSecurity**

anotasyonuna sahip bir konfigürasyon sınıfı olmalıdır

```
@EnableGlobalMethodSecurity(prePostEnabled = true,  
securedEnabled = true, jsr250Enabled = true)  
public class PetClinicSecurityConfiguration {  
    ...  
}
```

# Metot Düzeyinde Yetkilendirme

```
@Service
public class PetClinicServiceImpl implements PetClinicService
{

    @PreAuthorize("hasAnyRole('ROLE_USER','ROLE_EDITOR')")
    public List<Owner> findOwners() {
        return ownerRepository.findAll();
    }

    @PreAuthorize("hasRole('ROLE_EDITOR') and
                    hasIpAddress(192.168.1.0/24)")
    public void createOwner(Owner owner) {
        ownerRepository.create(owner);
    }

    ...
}
```

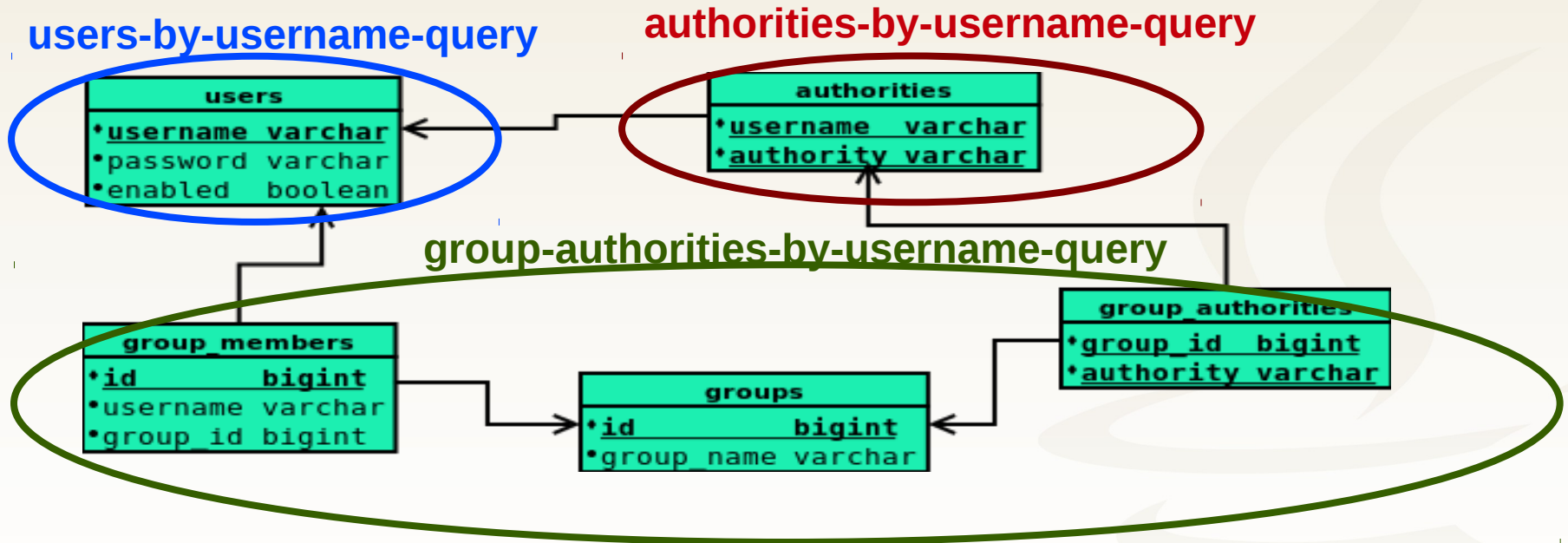
# Kullanıcı Bilgilerine JDBC ile Erişilmesi

- Spring Security'nin kimlik bilgileri **üç ayrı bölümde** ele alınabilir
  - Kullanıcı
  - Rol
  - Grup
- Bu üç ayrı olgunun bir birleri ile **M:N ilişkileri** de söz konusudur
  - Kullanıcı – Rol
  - Kullanıcı – Grup
  - Rol – Grup

# Kullanıcı Bilgilerine JDBC ile Erişilmesi

- Kimliklendirme sırasında veritabanından bu üç bölümle ilgili **şu bilgiler sorgulanır**
  - Kullanıcı adına göre **kullanıcı bilgisi**
    - Kullanıcı adı, şifre ve aktif/pasif bilgisi beklenir
  - Kullanıcı adına göre **rol bilgileri**
    - Rol adları beklenir
  - Kullanıcı adına göre **kullanıcının gruplarına atanmış rol bilgileri**
    - Grup bilgisi ve rol adları beklenir

# Kullanıcı Bilgilerinin Veritabanından Alınması



# Kullanıcı Bilgilerinin Veritabanından Alınması

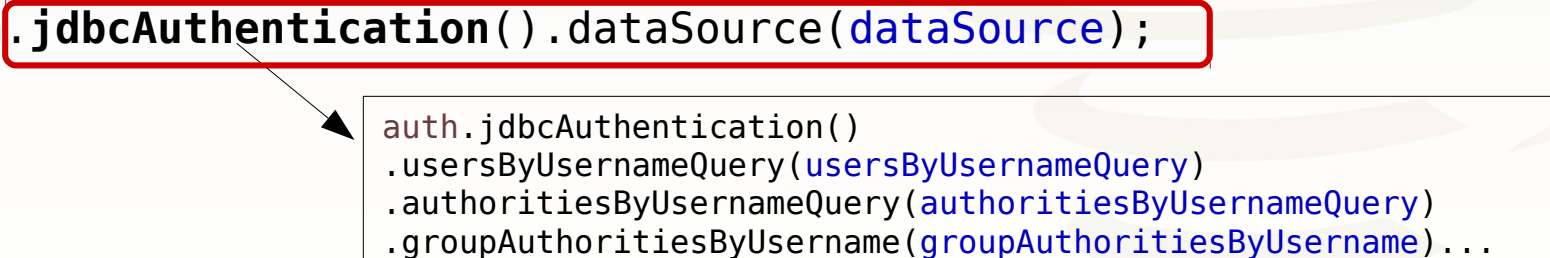
```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
{
    ...

    @Autowired
    private DataSource dataSource;

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {

        auth.jdbcAuthentication().dataSource(dataSource);

    }
}
```



```
auth.jdbcAuthentication()
    .usersByUsernameQuery(usersByUsernameQuery)
    .authoritiesByUsernameQuery(authoritiesByUsernameQuery)
    .groupAuthoritiesByUsername(groupAuthoritiesByUsername)...
```

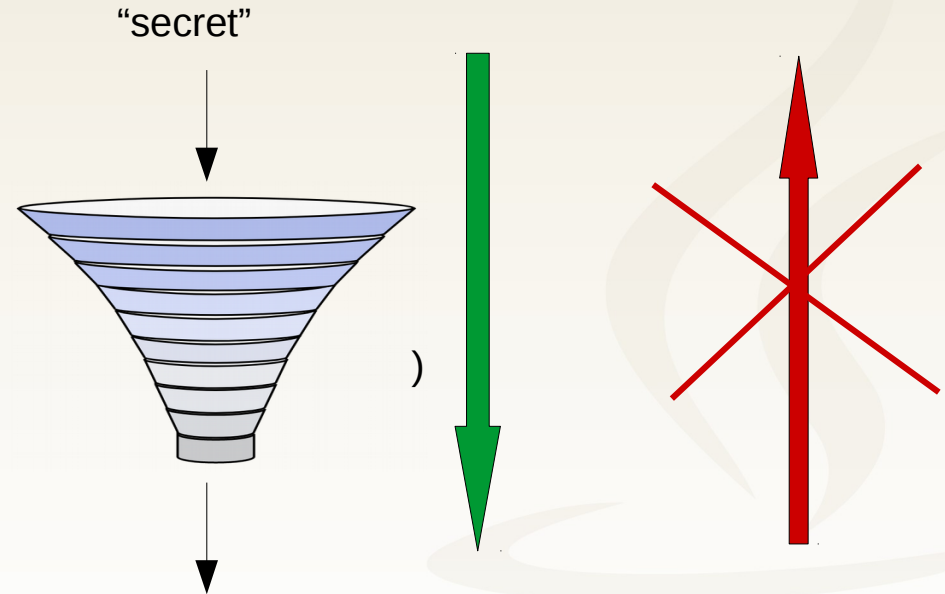


# Kirptolu Şifre Kullanımı

- Spring Security **şifrelerin** DB'de **kriptolu olarak** saklanmasını sağlar
- Şifreleri kriptolamak için **değişik algoritmalar** kullanılabilir
  - bcrypt, pbkdf2, scrypt, md4, sha...
- Bunların hepsi **tek yönlü** algoritmalarıdır

# Kirptolu Şifre Kullanımı

```
bcryptPasswordEncoder.encode(
```



```
$2a$10$0nz0StCmfnC9OGRSM0t7xOejjPO.Ytl2JzFUxp8HYubBAYtGUEAQK
```

# Bcrypt Nasıl Çalışır?

- Parola bilgilerini saklamaya yönelik tasarlanmış **tek yönlü bir “hash”** fonksiyonudur
- Diğer hash fonksiyonlarına kıyasla **çok daha yavaş** çalışır
- bcrypt()'in çalışması **100 ms** civarı sürer
- Bu süre **bir kullanıcı** açısından çok uzun değildir
- Ancak bu yavaşlık **topluca parolaların sınanmasını** oldukça zorlaştırır

# Bcrypt Nasıl Çalışır?

- Bu yavaş çalışmanın temelinde bir döngü içerisinde **defalarca dahili olarak başka bir hash fonksiyonunu çalıştırması** yatar
- Bu **döngünün sayısı** dışarıdan değiştirilebilir
- Bcrypt yavaş olduğu için CPU'ların hızlanması ile gündemden düşen “**rainbow tabloları**” yine ortaya çıkmıştır
- Bunun da önüne geçmek için bcrypt tarafından **kullanıcıya özgü rastgele salt değeri** kullanılmaktadır

# Bcrypt Nasıl Çalışır?

- Böylece “rainbow tabloları”nda tutulan sık kullanılan parolaların **önceden hazırlanmış hash değerleri** işlevsiz hale gelmektedir
- Her kullanıcı için **hash değerinin ayrı ayrı elde edilmesi** gerekir
- Salt değeri anlık **random bir değer** olarak elde edilmektedir
- Ayrıca bu salt değeri **hash değer ile birlikte** tutulmaktadır

# Bcrypt Nasıl Çalışır?

**\$2a\$12\$8vxYfAWCXe0Hm4gNX8nzwuqWNuk0kcMJ1a9G2tD71ipotEZ9f80Vu**

Bcrypt  
versiyonu

128 bit random  
salt değeri

184 bit hash değeri

İterasyon sayısı. En  
az 12 olmalı

# Bcrypt Nasıl Çalışır?

```
bcrypt(cost, salt, input)
  state := EksBlowfishSetup(cost, salt, input)
  ctext := "OrpheanBeholderScryDoubt"
  repeat (64)
    ctext := EncryptECB(state, ctext)
  return Concatenate(cost, salt, ctext)
```

# Kirptolu Şifre Kullanımı

- Kimliklendirme sırasında **kullanıcının girdiği şifre** algoritmaya göre kriptolanarak **DB'deki kriptolu değer** ile karşılaştırılır

| username | password  |
|----------|---|
| user1    | <u>{noop}</u> secret  |
| user2    | <u>{md4}</u> 67d3dafef63ff00603aeef3769cfbf0d                                   |
| user3    | <u>{bcrypt}</u> \$2a\$10\$0nz0StCmfnC9OGRSM0t7xOejjPO.Ytl2JzFUxp8HYubBAYtGUEAQK |



# Oturum Yönetimi

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        ...

        http.sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
            .maximumSessions(1)
            .maxSessionsPreventsLogin(false)
            .expiredUrl("/login");
    }

    @Bean
    public ServletListenerRegistrationBean<HttpSessionEventPublisher>
    httpSessionEventPublisher() {
        return new ServletListenerRegistrationBean<HttpSessionEventPublisher>(
            new HttpSessionEventPublisher());
    }
}
```

# Oturum Yönetimi

- Tomcat Web Container'ın session timeout değeri default 30 dk'dır
- Application.properties içerisinde değiştirilebilir

```
server.session.timeout=60
```

# İletişim

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

