

# Spring Boot Framework 1



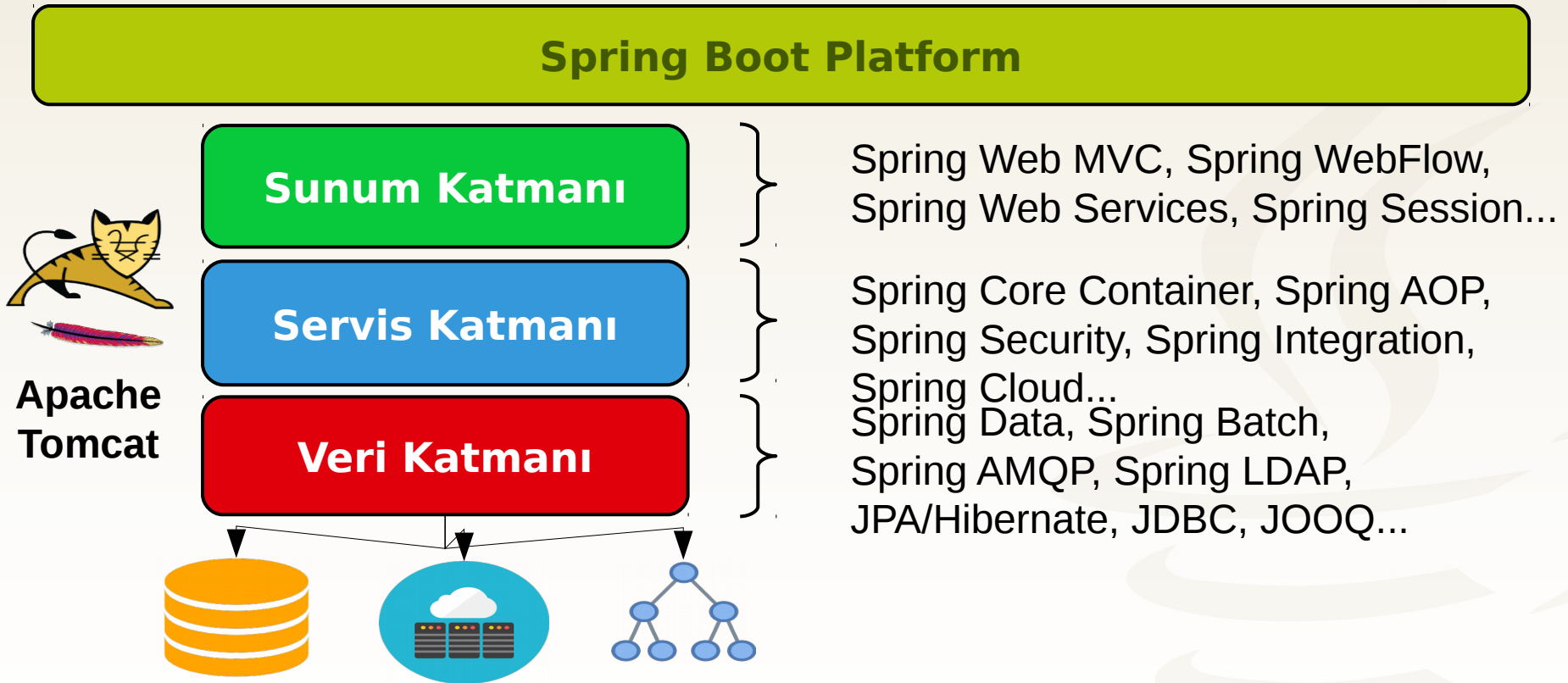
# Spring Boot Nedir?

- Spring ekosistemi üzerine kurulmuş bir **framework**tür
- Temel amacı Spring ekosistemindeki teknolojiler ile **çalışmayı hızlandırmaktır**
- Kurumsal uygulamaların ihtiyaç duyduğu pek çok **altyapısal servisi** hazır biçimde sunmaya çalışır
- Uygulamaların **konfigürasyonunu** ve **kurulumunu** kolaylaştırır

# Spring Boot Ne Değildir?

- Spring Boot bir **uygulama sunucusu veya web container değildir**, ama tomcat gibi web container'ları gömülü olarak çalıştırabilir
- **JSR spesifikasyonlarını implement etmez**, ama Hibernate gibi pek çok JSR implemantasyonunun kullanımını kolaylaştırır
- Hiçbir şekilde **otomatik kod üretmez**, konfigürasyonu otomatik hale getirir

# Spring Boot'un Spring Ekosistemindeki Yeri



# Spring Ekosistemi ve Spring Boot Arasındaki İlişki



**Spring Application (Core)  
Framework ve diğer frameworkler...**

# Spring Ekosistemi ve Spring Boot Arasındaki İlişki



**Spring Boot!**

# Spring Boot ile Çalışmaya Başlamak

- Spring Boot ile proje geliştirirken **Maven**, **Gradle** veya **Ant** build araçlarından herhangi birisi kullanılabilir
- Maven kullanarak simple bir **Java projesi** yaratılarak çalışmaya başlanabilir
- İhtiyaç duyulan Spring Boot kabiliyetlerini aktive etmek için ilgili **bağımlılıkları kademeli olarak ekleyerek ilerlemek** mümkündür

# Spring Boot Starters

- Spring Boot'un ilgili kabiliyetlerini, 3rd party bağımlılıkları ile birlikte içeren **bağımlılık tanımlarıdır**
- Spring Boot içerisindeki herhangi bir kabiliyeti devreye almak için çoğunlukla ilgili **starter tanımını pom.xml'e eklemek yeterli** olmaktadır

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

Spring Web MVC kabiliyetlerini  
ve ilgili bağımlılıkları devreye alır

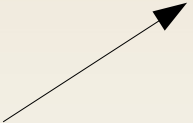


# Spring Boot Dev Tools

- Uygulama **geliştirme sürecini** kolay hale getirmek için de kabiliyetler sunar
  - Classpath'deki dosyalarda herhangi bir değişiklikte otomatik restart
  - Template engine gibi kısımlardaki önbellek kabiliyetlerini devre dışı bırakmak
  - Herhangi bir web kaynağında değişiklik olduğunda browser reload

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-devtools</artifactId>  
  <optional>true</optional>  
</dependency>
```

# İlk Spring Boot Uygulaması

 @SpringBootApplication, @ComponentScan ve @EnableAutoConfiguration anotasyonlarını bir araya getirir. Otomatik konfigürasyon sayesinde Spring ApplicationContext classpath'deki sınıflar ve bean tanımlarına göre otomatik oluşturulur

@SpringBootApplication

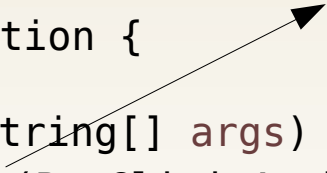
```
public class PetClinicApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(PetClinicApplication.class, args);
```

```
    }
```

```
}
```

 Uygulamayı JVM'de default ayarlarla çalıştırır

- Ardından IDE içerisinde Run As Spring Boot Application seçeneği ile çalıştırılarak <http://localhost:8080/actuator/health> adresine bir istek gönderilirse cevap şu şekilde olmalıdır: 

```
{"status": "UP"}
```

# Spring Boot Actuator

- Spring Boot, uygulamamızı çalışma zamanında yönetmek ve monitör etmek için bir takım kabiliyetler sunar
- Bu kabiliyetler **actuator** olarak adlandırılır

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

# Spring Boot Actuator Neler Sunar?

- Uygulama ile ilgili konfig ve durum bilgisi
- Çalışma zamanında log konfigürasyonu
- Uygulama ve sistem ile ilgili pek çok metrik bilgisi
  - Yüklenen sınıflar, heap size, thread count, veritabanı/cache erişimi, process sayısı, disk kapasitesi gibi
- Spring Boot Actuator servislerine erişmek için **HTTP endpoint**'leri, **JMX** veya **SSH/Telnet** gibi araçlar kullanılabilir

# Actuator Endpointlerinin Aktivasyonu

- Varsayılan durumda **bütün hizmetler aktiftir**
- İstenirse tek tek istenirse de bütün hizmetleri topluca **devre dışı** bırakmak mümkündür

src/main/resources/application.properties



```
management.endpoint.health.enabled=false  
management.endpoints.enabled-by-default=false
```

# Actuator Endpointlerine Web Erişimi

- Varsayılan durumda sadece **/health** ve **/info** servisleri web'den erişilebilir
- Diğer servislere de **web üzerinden erişim** izni verilebilir

src/main/resources/application.properties



```
management.endpoints.web.exposure.include=*
```

# Actuator Endpointlerine Web Erişimi

- Varsayılan durumda actuator servislerine erişmek için URI prefix'i olarak **/actuator** eklemek gerekir
- Bu **prefix** değiştirilebilir

src/main/resources/application.properties



```
management.endpoints.web.base-path=/actuator
```

# Spring Boot ve Logging

- Application.properties içerisinde logger'lar ve log düzeyleri, log çıktısının formatı, log dosyalarının lokasyonu vb yönetilebilir

```
logging.level.root=INFO
logging.level.org.hibernate.type=TRACE
logging.level.org.springframework.web=DEBUG
```

- Kullanılan log altyapısına göre ilgili **log konfigürasyon dosyaları** classpath'den yüklenebilir

```
logging.config=logback-spring.xml
```

logback-spring.xml,  
logback.xml,  
log4j2-spring.xml,  
log4j2.xml,  
logging.properties



# Spring Boot ve Logging

- Log çıktısı default **console**'a yazılır, ancak istenirse **log dosyalarına** da yazılabilir

```
logging.file=petclinic.log
logging.path=/tmp
```

logging.file tanımlı ise path devreye girmez, sadece path tanımlı olduğu durumda ise spring.log isimli log dosyası belirtilen dizinde oluşturulur

- Log property tanımları fiziksel log altyapısından bağımsızdır

# Konfigürasyon Property Tanımları

- Komut satırı parametreleri

- `$java -jar petclinic.jar --spring.thymeleaf.enabled=false`

- JVM sistem değişkenleri

- `$java -Dspring.thymeleaf.enabled=false -jar petclinic.jar`

- İşletim sistemi çevre değişkenleri

- `$set spring.thymeleaf.enabled=false`
- `$java -jar petclinic.jar`

- classpath'deki application.properties dosyası

↓  
`spring.thymeleaf.enabled=false`

- `$java -jar petclinic.jar`

# Application.properties'in Arandığı Yerler

- Spring Boot **application.properties** dosyasını sırası ile aşağıdaki lokasyonlarda arar
  - file:./config/
  - file:./
  - classpath:/config/
  - classpath:/

# Application.properties'in Özelleştirilmesi

- Spring Boot application.properties'in **isminin ve arandığı lokasyonların değiştirilmesine** izin verir
  - `spring.config.name=project`
  - `spring.config.location=classpath:/myconfig/`
  - `spring.config.location=classpath:/default.properties,classpath:/overriden.properties`

# Spring Profil Kabiliyeti ve Application.properties

- **application.properties** dosyaları **aktif Spring profillerine** göre de yüklenebilir
- Bunun için classpath'de **application-\${profile}.properties** şeklinde konfigürasyon property dosyaları yer almalıdır
- Bu dosyalar **application.properties sonrasında** yüklenmektedir
- Aktif profiller **-Dspring.profiles.active** sistem değişkeni ile tanımlanabilir

# Spring Profil Kabiliyeti ve Application.properties Örnek

```
$java -Dspring.profiles.active=dev,h2 -jar petclinic.jar
```

```
petclinic.jar  
|__application.properties  
|__application-dev.properties  
|__application-h2.properties
```

# Default Profil Kabiliyeti

- Eğer hiç aktif profil tanımı yoksa Spring “**default**” profil değerini aktive eder
- Böyle bir durumda, erişilebilir bir **application-default.properties** dosyası varsa yüklenir

# Konfigürasyon Ayarlarının YAML ile Yönetilmesi

- **YAML JSON**'ın superset'idir
- **Hiyerarşik konfigürasyon verisinin** tanımlanması için oldukça uygun bir formattır
- Spring Boot, **YAML formatını** da destekler



# YAML Örneği

application.yml

```
server:
  port: 8081
  context-path: /petclinic
  tomcat:
    basedir: /tmp
    max-connections: 1000

logging:
  file: petclinic.log
  config: classpath:/custom-log4j.xml
```

application.properties



```
server.port=8081
server.context-path: /petclinic
server.tomcat.basedir: /tmp
server.tomcat.max-connections: 1000

logging.file: petclinic.log
logging.config: classpath:/custom-log4j.xml
```

# Spring Profile Kabiliyeti ve YAML

- Tek bir YAML dosyasında **birden fazla Spring profili** için property tanımları yapılabilir

```
server:
  address: 192.168.1.1
---
spring:
  profiles: dev
server:
  address: 127.0.0.1
---
spring:
  profiles: prod
server:
  address: 107.180.27.155
```

# YAML ve Default Profil Kullanımı

- Tanımlı profillerden hiç birisi aktif değilse, varsa **default profil tanımı** devreye alınır

```
spring:
  profiles:default
server:
  port: 8081
---
spring:
  profiles:dev
server:
  port: 8082
---
spring:
  profiles:prod
server:
  port: 80
```

# Tip Güvenli Konfigürasyon Yönetimi

application.properties

```
petclinic.period=1000
petclinic.adminRoles=VET,ADMIN

petclinic.fileTransfer.enabled=true
petclinic.fileTransfer.downloadPath=/downloads
petclinic.fileTransfer.uploadPath=/uploads
```

@Component

```
public class FileTransferManager {

    @Value("${petclinic.fileTransfer.enabled}")
    private boolean fileTransferEnabled;

    @Value("${petclinic.fileTransfer.downloadPath}")
    private String downloadPath;

    @Value("${petclinic.fileTransfer.uploadPath}")
    private String uploadPath;

    ...
}
```

# Tip Güvenli Konfigürasyon Yönetimi

```
@ConfigurationProperties(prefix="petclinic")
public class PetClinicProperties {

    private long period;

    private List<String> adminRoles = new ArrayList<>();

    private FileTransfer fileTransfer = new FileTransfer();

    //getter & setters...

    public static class FileTransfer {

        private boolean enabled;

        private String downloadPath;

        private String uploadPath;

        // getter & setters...

    }
}
```

# Tip Güvenli Konfigürasyon Yönetimi

```
@Component
public class FileTransferManager {

    @Autowired
    private PetClinicProperties petClinicProperties;

    public void checkUploadsFolder() {
        if(petClinicProperties.getFileTransfer().isEnabled()) {
            //...
        }
    }
}
```

# ConfigurationProperties Sınıflarının Tanıtılması

- Konfigürasyon sınıfları **@EnableConfigurationProperties** ile tanıtılmalıdır

```
@EnableConfigurationProperties(value={PetClinicProperties.class})
```

```
@SpringBootApplication
```

```
public class PetClinicApplication {
```

```
...
```

```
}
```

# Spring Web MVC

- Spring Boot ile web uygulamaları ve REST servisleri geliştirmek için **web starter**'ının eklenmesi yeterlidir
- pom.xml'deki bu tanım **DispatcherServlet** ve diğer pek çok Web MVC özelliğini devreye sokar

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



# Server Port ve Context Path Ayarları

- Spring Boot web uygulamasını varsayılan durumda **8080** portunda ve / URI path'inden erişilecek biçimde deploy eder
- Bu ayarlar application.properties içerisinde değiştirilebilir

```
server.port=8081  
server.context-path=/petclinic
```

# DispatcherServlet

- DispatcherServlet default olarak bütün web isteklerini yakalayacak biçimde tanımlıdır
- **application.properties**'de bir tanım ile sadece belirli web isteklerini ele alması sağlanabilir

```
server.servlet-path=/mvc/*
```

# Statik Web Resource'larına Erişim

- Eğer bir web isteği **@RequestMapping** ile eşleşmemiş ise **statik web resource** olarak çözülür
- Bunun için default olarak sırası ile classpath'deki **META-INF/resources**, **resources**, **static** veya **public** dizinlerine, yada **ServletContext root** dizinine bakılır
- İstenirse bu lokasyonlar `application.properties`'den özelleştirilebilir

```
spring.resources.static-locations=classpath:/webresources
```

# Statik Web Resource'larına Erişim

- Default olarak statik resource'lara yapılan istekler `/**` örüntüsü ile eşleştirilir
- İstenirse bu `application.properties`'deki tanımla değiştirilebilir

```
spring.mvc.static-path-pattern=/resources/**
```

- Spring Boot ayrıca webjars formatındaki jar dosyalarının içerisindeki resource'lara **`/webjars/**`** örüntüsü ile erişim imkanı da sunar

# Index.html ve Favicon

- Spring Boot **index.html**'i statik web resource'larının bulunduğu lokasyonlardan sunmaktadır
- **favicon.ico** dosyasını da belirtilen lokasyonlardan ve root classpath'den aramaktadır
- Eğer bu dosyalar mevcut ise uygulamanın **default home sayfası** ve **favicon ikonu** olarak kullanılırlar

# Spring Boot ve JSP

- Web container'ların **executable JAR** içerisinde yer alan JSP sayfalarına erişiminde sorunlar olabilmektedir
- Spring Boot'un önerisi dinamik web sayfaları için **JSP yerine** Thymeleaf, FreeMarker gibi **template engine'ler** kullanılmasıdır
- Ancak web container olarak **Tomcat kullanıyorsanız** küçük bir ayarlama sonrası JSP teknolojisi ile rahatlıkla çalışabilirsiniz

# JSP Kullanmak İçin Yapılması Gerekenler

- JSP sayfaları **src/main/webapp** dizini altında bir lokasyonda oluşturulmalıdır
- Dolayısı ile projenin packaging tipi **war** olmalıdır

```
<packaging>war</packaging>
```

- Tomcat ile çalışırken **jasper compiler**'ın classpath'de yer alması gerekir

```
<dependency>  
  <groupId>org.apache.tomcat.embed</groupId>  
  <artifactId>tomcat-embed-jasper</artifactId>  
</dependency>
```

# JSP Kullanmak İçin Yapılması Gerekenler

- JSP sayfaları içerisinde JSTL vs kullanılıyor ise **JSTL kütüphanesi** projenin classpath'inde yer almalıdır

```
<dependency>  
    <groupId>javax.servlet</groupId>  
    <artifactId>jstl</artifactId>  
</dependency>
```

- JSP sayfalarını çözümleyen **ViewResolver**'a prefix ve suffix olarak aşağıdaki değerler verilebilir

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



# Spring Boot ve JSP Kısıtları

- Undertow server, JSP sayfalarını **desteklememektedir**
- **error.jsp** şeklinde bir sayfa default error view yerine geçmeyecektir

# Spring Boot ve Template Kullanımı

- Spring Boot view teknolojisi olarak JSP yerine **template engine**'ler ile çalışmayı önermektedir
- Dinamik web sayfaları için default olarak aşağıdaki template engine'ler için **otomatik konfigürasyon imkanı** sunar
  - Thymeleaf
  - FreeMarker
  - Groovy
  - Mustache

# Spring Boot ve Template Kullanımı

- Bunların ilgili **starter**'ını eklemek yeterlidir

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

- Default olarak template dosyaları **classpath:/templates** isimli bir lokasyondan yüklenir
- İstenirse bu lokasyon ve suffix'ler özelleştirilebilir

```
spring.thymeleaf.prefix=classpath:/templates/  
spring.thymeleaf.suffix=.html
```

# Spring Boot ve Template Kullanımı

- Şablon motoru ile ilgili starter eklendiğinde **view resolution kabiliyeti** de otomatik olarak devreye girer
- Starter tanımını pom.xml'den çıkarmadan şablonun view resolution kabiliyeti **devre dışı** bırakılabilir

```
spring.thymeleaf.enabled=true
```

# Şablonların Ön Bellekten Erişilmesi

- Şablon dosyalarının içeriği **ön bellekte** tutulabilmektedir
- Bu davranış şablon motorunun varsayılan tanımına göre **devre dışı** bırakılabilir veya **devreye** alınabilir

```
spring.thymeleaf.cache=true
```

# Spring Boot ve Hata Sayfaları

- HTTP statü kodlarına karşılık gelen **custom hata sayfaları** oluşturmak için **statik web resource lokasyonları altında /error** isimli bir dizin oluşturulmalıdır
- Bu dizin altında **statü koduna karşılık gelen html sayfa** oluşturulmalıdır
- Hata sayfaları **dinamik içeriğe** de sahip olabilir
- Bu durumda /error dizini **classpath:/templates** altında yer almalıdır

# Default White Label Hata Sayfası

- Eğer statü koduna karşılık gelen bir hata sayfası mevcut değil ise Spring Boot **whitelabel hata sayfası** çıkarır

## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Jun 04 16:48:11 EET 2018

There was an unexpected error (type=Not Found, status=404).

No message available

# Servlet, Filter veya EventListener Tanımlamak

```
@WebServlet(urlPatterns="/pcs")
public class PetClinicServlet
    extends HttpServlet {

    @Override
    protected void doGet(
        HttpServletRequest req,
        HttpServletResponse resp)
        throws ServletException,
        IOException {
        resp.getWriter()
            .write("pcs...");
    }
}
```

```
@WebFilter(urlPatterns="/*")
public class PetClinicFilter
    implements Filter {

    ...

    @Override
    public void doFilter(ServletRequest
        arg0, ServletResponse arg1, FilterChain
        arg2)
        throws IOException,
        ServletException {
        try {
            System.out.println("before...");
            arg2.doFilter(arg0, arg1);
        } finally {
            System.out.println("after...");
        }
    }
}
```



# Servlet, Filter veya EventListener Tanımlamak

```
@WebListener
public class PetClinicHttpSessionListener implements HttpSessionListener {

    @Override
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("Session created :" + se.getSession().getId());
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("Session destroyed :" +
se.getSession().getId());
    }
}
```

```
@SpringBootApplication
@ServletComponentScan
public class PetClinicApplication {
    ...
}
```

# Servlet, Filter veya EventListener Tanımlamak

- 3rd party sınıflarda @WebServlet, @WebFilter veya @WebListener anotasyonları **kullanılmamış** olabilir
- Bu gibi durumlarda konfigürasyon sınıfı içerisinde **ServletRegistrationBean, FilterRegistrationBean, veya ServletListenerRegistrationBean** sınıfları ile aynı şekilde servlet, filter veya listener tanımlanabilir

# Servlet, Filter veya EventListener Tanımlamak

@Configuration

```
public class PetClinicConfiguration {  
    @Bean  
    public ServletRegistrationBean pcs() {  
        return new ServletRegistrationBean(  
            new PetClinicServlet(), "/pcs");  
    }  
  
    @Bean  
    public FilterRegistrationBean pcsFilter() {  
        FilterRegistrationBean bean = new FilterRegistrationBean(  
            new PetClinicFilter());  
        bean.setUrlPatterns(Arrays.asList("/*"));  
        bean.setOrder(Ordered.HIGHEST_PRECEDENCE);  
        return bean;  
    }  
  
    @Bean  
    public ServletListenerRegistrationBean pcsListener() {  
        return new ServletListenerRegistrationBean<>(  
            new PetClinicHttpSessionListener());  
    }  
}
```

# Statik Web Resource'ları ve Cache Busting

- CSS, JS gibi statik resource'lar **tarayıcı tarafında cache**'lenerek müteakip isteklerde sunucuya erişmeden kullanıcıya hızlıca gösterilebilirler
- Geliştirme ortamında bu özellik, ilgili dosyalarda yapılmış **değişikliklerin görüntülenmesini** engelleyecektir
- Spring Boot bu problemi aşmak için **cache busting** kabiliyeti sunar

# Statik Web Resource'lar ve Cache Busting Örnek

index.html

```
<head>  
  <link rel="stylesheet" type="text/css" th:href="@{/css/petclinic.css}"/>  
</head>
```

yüklenirken...

```
<head>  
  <link rel="stylesheet" type="text/css" href="css/petclinic-  
2a2d595e6ed9a0b24f027f2b63b134d6.css"/>  
</head>
```

# Cache Busting'in Devreyeye Alınması

- **Devreye almak için**  
application.properties'de ilgili property tanımı yapılmalıdır

```
spring.resources.chain.strategy.content.enabled=true  
spring.resources.chain.strategy.content.paths=/**
```

# Cache Busting'in Devreye Alınması

- Cache Busting'i arka planda gerçekleştiren **ResourceUrlEncodingFilter**'dir
- Spring Boot **Thymeleaf** ve **FreeMarker** template engine'leri devreye alındığında bu filter'ı da **otomatik** aktive eder
- **JSP** veya **diğer template engine**'ler için ResourceUrlEncodingFilter'ın projeye **manuel** biçimde dahil edilmesi gerekir

# ResourceUrlEncodingFilter'in Manuel Konfigürasyonu

```
@Configuration  
public class FilterConfiguration {
```

```
    @Bean  
    public FilterRegistrationBean resourceUrlEncodingFilter() {  
        FilterRegistrationBean bean = new FilterRegistrationBean(  
            new ResourceUrlEncodingFilter());  
        bean.setUrlPatterns(Arrays.asList("/*"));  
        bean.setOrder(Ordered.HIGHEST_PRECEDENCE);  
        return bean;  
    }
```

```
}
```



# JS Dosyaları ve Cache Busting

- Angular, Bootstrap, JQuery gibi **JS teknolojileri** ile çalışırken ise JS dosyalarında resource path'e **statik versiyon prefix'i** eklenir

```
spring.resources.chain.strategy.fixed.enabled=true  
spring.resources.chain.strategy.fixed.paths=/js/lib/  
spring.resources.chain.strategy.fixed.version=v12
```

# JS Dosyaları ve Cache Busting Örnek

```
$.getScript('/js/lib/mymodule.js', function()  
{  
    window.alert('my module loaded!');  
});
```

yüklenirken...

```
$.getScript('/v12/js/lib/mymodule.js', function()  
{  
    window.alert('my module loaded!');  
});
```

# Spring Boot ve REST

- Spring Boot'un çıkış hedeflerinden birisi de mikroservis mimarisine uygun **headless web uygulamaları** geliştirmektir

```
@RestController
public class PetClinicRestController {

    @RequestMapping("/hello")
    public String welcome() {
        return "Welcome to PetClinic WebApp!";
    }
}
```

# RestTemplate Konfigürasyonu

- Uzaktaki REST servislerini çağırmak için **RestTemplate** sınıfı kullanılabilir
- RestTemplate nesneleri kullanım yerlerine göre farklı ayarlarla oluşturulduklarından Spring Boot **genel amaçlı bir RestTemplate bean'i sunmaz**
- Bunun yerine **RestTemplateBuilder** bean'i sunarak RestTemplate'in konfigürasyonu kullanılan yerde yapılabilir

# RestTemplateBuilder Kullanımı

- **RestTemplateBuilder sayesinde, örneğin gerekli **HttpMessageConverter** nesneleri, HTTP basic auth vs RestTemplate instance'larına kolayca uygulanır**

```
@Service
public class PersonService {

    private RestTemplate restTemplate;

    @Autowired
    public PersonService(RestTemplateBuilder restTemplateBuilder) {
        restTemplate = restTemplateBuilder
            .basicAuthorization("user1", "secret")
            .defaultMessageConverters()
            .build();

        ...
    }
}
```

# Response Content Tipinin Belirlenmesi

- Spring REST servis çağrılarında istemci sunucudan **resource'u hangi formatta dönmesi gerektiğini** aşağıdaki yöntemlerden birisi ile belirtebilir
  - **Request URI**'inin uzantısı
  - **format** isimli bir request parametresinin değeri
  - **Accept Request Header**'i
- Spring Boot **default olarak** sadece “accept request header”ı devreye alır

# Request URI'nin Uzantısı

- Devreye almak için application.properties içerisinde aşağıdaki property tanımları yapılmalıdır

```
spring.mvc.contentnegotiation.favor-path-extension=true  
spring.mvc.pathmatch.use-suffix-pattern=true
```

# Format Request Parametresi

- Devreye almak için application.properties içerisinde aşağıdaki property tanımı yapılmalıdır

```
spring.mvc.contentnegotiation.favor-parameter=true
```



# Spring Boot ve HATEOAS

- RESTful API geliştirenler için Spring **HATEOAS**'ın **otomatik konfigürasyonunu** yapar
- Aktive etmek için hateoas starter'ı pom.xml'e eklenmelidir

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

# HATEOAS'a Uygun REST Servis Cevabı

```
@RestController
public class PetClinicRestController {
    @Autowired
    private PetClinicService petClinicService;
    @RequestMapping(method = RequestMethod.GET, value = "/owner/{id}")
    public ResponseEntity<?> getOwner(@PathVariable("id") Long id) {
        try {
            Owner owner = petClinicService.findOwner(id);

            ControllerLinkBuilder linkToController =
                ControllerLinkBuilder.linkTo(PetClinicRestController.class);
            Link self = linkToController.slash("/owner/" + id).withSelfRel();
            Link create = linkToController.slash("/owner").withRel("create");
            Link update = linkToController.slash("/owner/" + id).withRel("update");
            Link delete = linkToController.slash("/owner/" + id).withRel("delete");

            Resource<Owner> resource = new Resource<Owner>(owner,
                                                            self, create, update, delete);

            return ResponseEntity.ok(resource);
        } catch (OwnerNotFoundException ex) {
            return ResponseEntity.notFound().build();
        }
    }
    ...
}
```

# HATEOAS'a Uygun REST Servis Cevabı

GET /owner/1



```
{
  "firstName": "Kenan",
  "lastName": "Sevindik",
  "_links": {
    "self": {"href": "http://localhost:8080/owner/1"},
    "create": {"href": "http://localhost:8080/owner"},
    "update": {"href": "http://localhost:8080/owner/1"},
    "delete": {"href": "http://localhost:8080/owner/1"}
  }
}
```

# Spring MVC Özelleştirmeleri

- Spring Boot, Spring MVC ile ilgili pek çok özelliği otomatik olarak devreye almaktadır
- Bazı durumlarda bu konfigürasyona uygulama spesifik ilaveler, örneğin interceptor, formatter, view controller, yapmak gerekebilir
- Bunun için WebMvcConfigurerAdapter sınıfından türeyen bir konfigürasyon sınıfı yazmak yeterlidir

```
@Configuration
public class CustomMvcConfig extends WebMvcConfigurerAdapter {
    ...
}
```

# Spring MVC Özelleştirmeleri

- Spring MVC konfigürasyonunu tamamen manuel yapmak, Spring Boot'un otomatik konfigürasyonunu devreden çıkarmak için konfigürasyon sınıfının üzerine **@EnableWebMvc** anotasyonu da eklenmelidir

```
@Configuration
```

```
@EnableWebMvc
```

```
public class CustomMvcConfig extends WebMvcConfigurerAdapter {  
    ...  
}
```

# İletişim

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

