Spring RequestMapping, PathVariable, RequestBody Ve ResponseBody Annotasyonları

@RequestMapping annotasyonu `/appointments` gibi **URL'lerin** bir **sınıf** veya **metod** tarafından **map** edilmesini sağlar. Sınıf üzerinde kullanıldığı zaman, o sınıfın belirtilen **URL** ile ilgili tüm işleri yapması sağlanır. Metod üzerinde kullanıldığı zaman daha spesifik **URL'ye** göre işlem yapılması sağlanmış olur.

```
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {

    private final AppointmentBook appointmentBook;

    @Autowired
    public AppointmentsController(AppointmentBook appointmentBook) {
        this.appointmentBook = appointmentBook;
    }

    @RequestMapping(method = RequestMethod.GET)
    public Map<String, Appointment> get() {
        return appointmentBook.getAppointmentsForToday();
    }

    @RequestMapping(value="/{day}", method = RequestMethod.GET)
    public Map<String, Appointment> getForDay(@PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Model
model) {
        return appointmentBook.getAppointmentsForDay(day);
    }

    @RequestMapping(value="/new", method = RequestMethod.GET)
    public AppointmentForm getNewForm() {
        return new AppointmentForm();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String add(@Valid AppointmentForm appointment, BindingResult result) {
        if (result.hasErrors()) {
            return "appointments/new";
        }
        appointmentBook.addAppointment(appointment);
        return "redirect:/appointments";
    }
}
```

get() metodu `/annotations` şeklinde **GET** request olduğu zaman çalışır.
getForDay() metodu `/annotations/12` gibi sayısal gün değeri verildiği zaman çalışır.
getNewForm() metodu `/annotations/new` şeklinde **GET** request olduğu zaman çalışır.
add() metodu `/annotations` şeklinde **POST** request olduğu zaman çalışır.

Görüldüğü gibi **GET** veya **POST** request'ine göre belirli metodların da çalışabilmesi sağlanmaktadır.

Sınıf adı üzerinde **@RequestMapping** yoksa, tüm path'ler **relative** yerine **absolute** olur. Yukarıdaki örnekte **relative** path kullanılmıştır.

```
@Controller
public class ClinicController {

    private final Clinic clinic;

    @Autowired
    public ClinicController(Clinic clinic) {
        this.clinic = clinic;
    }

    @RequestMapping("/")
    public void welcomeHandler() {
    }

    @RequestMapping("/vets")
    public ModelMap vetsHandler() {
        return new ModelMap(this.clinic.getVets());
    }

}
```

Not: welcomeHandler() metodu site ismiyle direkt erişildiği zaman çalışacaktır. Eğer site adı `example.com` ise bu metod çalışmış olur.

Template Pattern Kullanımı

```
@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable String ownerId, Model model) {
    Owner owner = ownerService.findOwner(ownerId);
    model.addAttribute("owner", owner);
    return "displayOwner";
}
```

@RequestMapping annotasyonundaki { } parantezleri arasındaki değer ile, **@PathVariable** annotasyonundaki değişken adı aynı olmalıdır. Bu örnekte iki değer de **ownerId'dir**

Not: Intellij **IDE** kullananlarda farklı isim girilirse altı çizili yazarak uyarı verilir

Birden fazla { } küme parentezi olduğu durumda ise, birden fazla **@PathVariable** tanımlanmalıdır:

```
@RequestMapping(value="/owners/{ownerId}/pets/{petId}", method=RequestMethod.GET)
public String findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
    Owner owner = ownerService.findOwner(ownerId);
    Pet pet = owner.getPet(petId);
    model.addAttribute("pet", pet);
    return "displayPet";
}
```

`{ownerId}` ve `{petId}` değerleri için **findPet()** metodu içerisinde iki tane parametre kullanılmıştır:

1. **@PathVariable** String ownerId
2. **@PathVariable** String petId

Not: ownerId ve petId parametrelerin türü String olarak tanımlandı. Bunun yerine temel tipler(**int,** double, Date vs.) kullanılabilirdi. Spring otomatik olarak en uygun olan tipe dönüştürür. Dönüştüremediği zaman **TypeMismatchException** hatası fırlatır.

Şu tarz kullanımda mümkündür:

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

    @RequestMapping("/pets/{petId}")
    public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
        // implementation omitted
    }

}
```

Bu örnekte görüldüğü gibi sınıf isminin üzerinde belirtilen **@RequestMapping** annotasyonu ile **relative path** yapılmıştır. Bu sınıf `/owners/{ownerId}` path'i ve bu path'in sonuna eklenecek path'lerin bu sınıfta ele alınmasını sağlar. Örneğin: `example.com/owners/42/` ve `example.com/owners/42/pets/32` gibi url'ler bu sınıf tarafından handle edilecektir.

Ayrıca, bir önceki örnekte görüldüğü gibi **findPet()** metodu aynı parametrelere sahip olmuştur.

Matrix Değişkenler

Diyelim ki `example.com/cars;color=red;year=2012` şeklinde **URL'lerde** her noktalı virgülle ayrılan `name-value` çiftlerine matrix değişkenler denir.

1. URL: `example.com/pets/42;q=11;r=22`

```
@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET)
public void findPet(@PathVariable String petId, @MatrixVariable int q) {

    // petId == 42
    // q == 11

}
```

@RequestMapping'de `"/"` ile kullanılan path atanır. Örneğimizde `"/"` şeklinde kullanılan path `/pets/42` dir. Metodun parametrelerine bakacak olursak, **@MatrixVariable** isminde yeni bir annotasyonun kullanıldığını görürürüz. İşte bu annotasyon `";"` ile belirtilen değişkenleri temsil eder. `";q=11"`

şeklinde tanımlanan path değerini **@MatrixVariable** **int** q ile temsil ettik. q değeri integer bir değer olduğu için String türü yerine **int** türü kullanıldı. Fakat `";r=22"` path değerini metod içerisinde kullanamayız. Çünkü metod parametresinde sadece `"q"` değişkeni tanımlanmıştır. Kullanmak için `"q"` değişkeni gibi tanımlamak gereklidir: `@MatrixVariable int r`

Son hali şu şekilde olur:

```
@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET)
public void findPet(@PathVariable String petId, @MatrixVariable int q, @MatrixVariable int r) {
    // petId == 42
    // q == 11
    // r==22
}
```

2. URL: `example.com/owners/42;q=11/pets/21;q=22`

pathVar attribute ile matrix değişkenlerinin yerini belirtmek gereklidir. Bu örneğimizde görüldüğü gibi q1 ownerId den sonra geldiği için **pathVar="ownerId"** değerini almıştır:

```
@RequestMapping(value = "/owners/{ownerId}/pets/{petId}", method = RequestMethod.GET)
public void findPet(
        @MatrixVariable(value="q", pathVar="ownerId") int q1,
        @MatrixVariable(value="q", pathVar="petId") int q2) {

    // q1 == 11
    // q2 == 22

}
```

3. URL: `example.com/pets/42`

Bir adresin hem normal şekilde hem de matrix değişkeni ile kullanılabilir olmasını sağlamak için **@MatrixVariable** annotasyonunun defaultValue değişkenine ilk değer atamamız gerekmektedir:

```
@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET)
public void findPet(@MatrixVariable(required=false, defaultValue="1") int q) {
    // q == 1
}
```

4. URL: `example.com/owners/42/pets/21;q=22;s=23`

Map sınıfını kullanarak tüm değerler alınabilir:

```
@RequestMapping(value = "/owners/{ownerId}/pets/{petId}", method = RequestMethod.GET)
@ResponseBody
public String findPet(@PathVariable String ownerId,@PathVariable String petId,
        @MatrixVariable Map<String, LinkedList<String>> matrixVars,
        @MatrixVariable(pathVar="petId") Map<String, String> petMatrixVars) {

    return "q = "+matrixVars.get("q").get(0)+"s = "+matrixVars.get("s").get(0);
}
```

Ekran çıktısı:

```
q = 22
s = 23
```

`Map<String,String>` değişkeninde **key** değeri `q, r ve s` değerlerinden biri olurken, value değeri **LinkedList** türünden olur.

Not: Matrix değişkenlerini kullanmak için spring konfigürasyon dosyasında `<mvc:annotation-driven enable-matrix-variables="true"/>` elementini eklemek gereklidir. Bu element Spring **3.2** versiyonu ile birlikte geldiği için daha alt versiyonlarda çalışmaz.

Regular Expressions Kullanımı

Bazı durumlarda, daha özel **URI** ile oluşmak gerekebilmektedir. Örneğin `"/spring-web/spring-web-3.0.5.jar"` şeklinde bir **URI** olsun. Bu **URI'yi** birden çok kısma ayırabilmek için Spring **MVC** regular expression kullanımına olanak tanımıştır.

`{varName:regex}` ifadesinde ilk kısım değişken adını ve ikinci kısım regular expression'u temsil eder.

```
@RequestMapping("/spring-web/{symbolicName:[a-z-]}-{version:\\d\\.\\d\\.\\d}{extension:\\.[a-z]}")
    public void handle(@PathVariable String version, @PathVariable String extension) {
        // ...
    }
}
```

Media Tip Kullanımı

consumes:

@RequestMapping annotasyonunun bir başka özelliği **consumes** tipidir. Bu özellik sayesinde belirli request header bilgisine göre spesifik bir metodun çalışması sağlanır.

```
@RequestMapping(value = "/consume", method = RequestMethod.GET, consumes = "image/jpeg")
@ResponseBody
public String consumes() {
    return "Only not text/plain header";
}
```

Bu örnekte, **consumes()** metodu sadece Content Type request header image/jpeg olduğu zaman çalışır. image/jpeg yerine text/html demiş olsa idik sadece text/html request header'a sahip olan request handle edilirdi.

!text/plain şeklinde bir kullanım yapıldığında, **text/plain** request header dışındaki diğer tüm request header türlerinde bu metod çalışır.

Eğer birden fazla **consumes** tipi kullanmak istiyorsak şu şekilde kullanmalıyız: `consumes = {"text/plain", "application/*"}`

produces:

@RequestMapping annotasyonunun bir başka özelliği **produces** tipidir. Bu özellik sayesinde metod **produces** değerine göre **Content-Type** dönderir. Örneğin, ajax request işlemlerinde kullanılan **application/json Content-Type** header bilgisi dönderebiliriz.

```
@RequestMapping(value = "/produces", method = RequestMethod.GET
,produces = "application/json")
@ResponseBody
public String consumes() {
    return "application/json";
}
```

Eğer birden fazla **produces** tipi kullanmak istiyorsak şu şekilde kullanmalıyız: `produces = {"text/plain", "application/*"}`

Request ve Header Parametre Kullanımı

You can narrow request matching through request parameter conditions such as "myParam", "!myParam", or **"myParam=myValue".** The first two test for request parameter presence/absence and the third for a specific parameter value. Here is an example with a request parameter value condition:

params kullanımı üç şekilde olabilmektedir: `myParam, !myParam veya myParam = myValue` .

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

    @RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET, params="myParam=myValue")
    public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
        // implementation omitted
    }

}
```

headers kullanımı üç şekilde olabilmektedir: `myHeader, !myHeader veya myHeader = myValue` .

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

    @RequestMapping(value = "/pets", method = RequestMethod.GET, headers="myHeader=myValue")
    public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
        // implementation omitted
```

```
    }

}
```

Desteklenen Metod Parametreleri ile ilgili makaleye erişmek için **tıklayınız** (/tutorial/Spring-RequestMapping-Supported-Method-Arguments)
Desteklenen Metod Dönüş Tipleri ile ilgili makaleye erişmek için **tıklayınız** (/tutorial/Spring-RequestMapping-Supported-Method-Return-Types)

@ResponseBody Kullanımı

@ResponseBody annotasyonu ile **String,** `application/json` veya `application/xml` türü gibi birden çok türden değerler dönderebiliriz.

Text(String) Yanıtı Döndermek

```
@RequestMapping(value = "/produceString", method = RequestMethod.GET)
@ResponseBody
public String produceString() {
    return "Hello World";
}
```

String değer döndermek için, metodun dönüş tipi String olmalıdır.

Not: Content-Type değeri `text/html` olur.

Json Yanıtı Döndermek

Spring **MVC** otomatik olarak bir nesneyi application/json formatına dönüştürebilmektedir. Bunun için;

1. Spring konfigürasyon xml dosyasında `mvc:annotation-driven` aktif hale getirilmelidir.
2. Bir **POJO** sınıfından nesne üretilmelidir. **POJO** sınıfı demek, getter ve setter metodlarına ve **default** constructor'a sahip olan sınıf demektir. Default constructor parametresiz constructor anlamına gelir.
3. Jakson kütüphanesini classpath'e eklenmelidir.
4. Bir metod uygun bir şekilde **map** edilmelidir.

`mvc:annotation-driven` aktif hale getirmek için aşağıdaki gibi spring config dosyasına eklenmelidir:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="mucayufa"/>
    <mvc:annotation-driven />

    <bean
            class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

Bu işlem tamamlandıktan sonra şimdi basit bir **POJO** sınıfı yaratalım:

```
public class User {
    private String userName;
    private String[] address;
    public User(){

    }
    public User(String userName, String[] address) {
        this.userName = userName;
        this.address = address;
    }

    public void setUserName(String userName) {
        this.userName = userName;
```

```
    }

    public void setAddress(String[] address) {
        this.address = address;
    }

    public String getUserName() {
        return userName;
    }

    public String[] getAddress() {
        return address;
    }

}
```

Not: Default constructor ve getter/setter metodları eklendi

`Jakson kütüphanesini ekleyelim:`

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.10</version>
</dependency>
```

`Son olarak bir metodu map edelim:`

```
@RequestMapping(value = "/produceJson", method = RequestMethod.GET)
@ResponseBody
public User produceJson() {
     User user =new User();
     user.setUserName("musonyan");
     user.setAddress(new String[]{"Ankara","İstanbul"});
     return user;
}
```

Görüldüğü gibi metodtan bir **User** nesnesi **return** ediliyor. Ayrıca **@ResponseBody** annotasyonu var. Spring otomatik olarak bu dönen değeri application/json formatına dönüştürür. Browserde `/produceJSon` **URI'sini** girdiğimiz zaman ekrana aşağıdaki gibi bir sonuç çıkacaktır:

```
{"userName":"musonyan","address":["Ankara","İstanbul"]}
```

Not: Content-Type değeri `application/json` olur.

XML yanıtı döndermek

1. Spring config dosyasına `mvc:annotation-driven` eklenir.
2. JAXB jar dosyası classpath'e eklenir. Maven kullanıyorsanız **JAXB** dependency'i pom.xml dosyasına eklenir.

```
<dependency>
    <groupId>javax.xml</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.1</version>
</dependency>
```

JAXB kütüphanesi eklendiği zaman Spring otomatik olarak **Jaxb2RootElementHttpMessageConverter** sınıfını context'e ekler.

3. @XmlRootElement ve **@XmlElement** annotasyonlarını kullanan bir **POJO** sınıfı yaratılır.

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import java.io.Serializable;

@XmlRootElement
public class Person implements Serializable {
    private int id;
    private String firstName;
    private String lastName;

    public Person() {
    }

    @XmlElement
    public int getId() {
```

```
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @XmlElement
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    @XmlElement
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import java.util.List;

@XmlRootElement(name = "people")
public class People {
    private List<Person> people;

    //Default constructor olmak zorunda
    public People() {
    }

    public People(List<Person> people) {
        this.people = people;
    }

    //getter olmak zorunda
    @XmlElement(name = "person")
    public List<Person> getPeople() {
        return people;
    }
    public void setPeople(List<Person> people) {
        this.people = people;
    }

}
```

4. Son olarak bir metod **map** edilir

```
@RequestMapping(value = "/produceXML", method = RequestMethod.GET)
@ResponseBody
public People produceXML() {
    Person aPerson = new Person();
    aPerson.setId(1);
    aPerson.setFirstName("Ahmet");
    aPerson.setLastName("Can");

    Person bPerson = new Person();
    bPerson.setId(2);
    bPerson.setFirstName("Ayşe");
    bPerson.setLastName("Yılmaz");

    return new People(Arrays.asList(aPerson,bPerson));
}
```

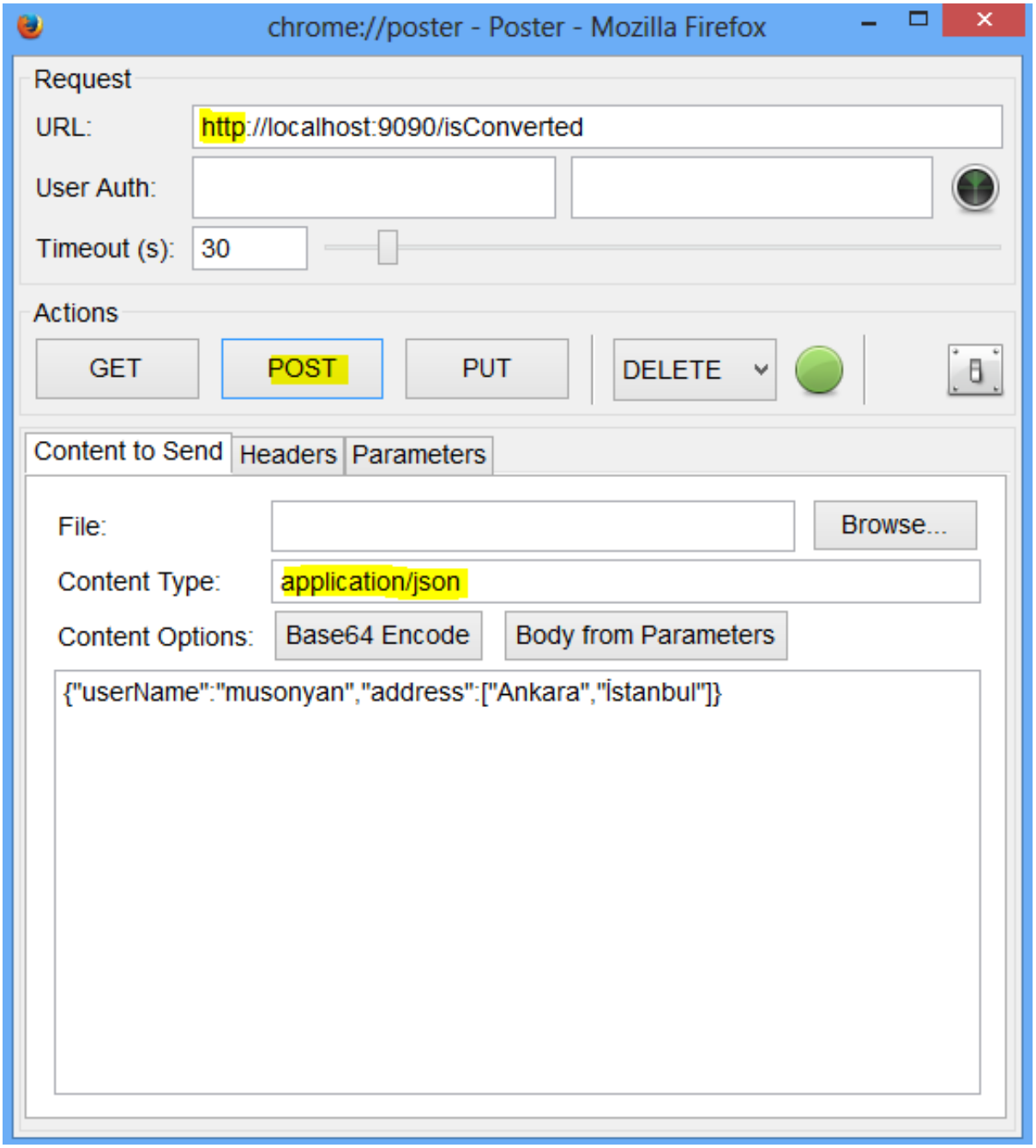
@RequestBody Annotasyonu

@RequestBody annotasyonu ile **POST** veya **PUT** request'leri handle edilir. Genelde **JSON** veya **XML** formatında bir request'i nesneye dönüştürmek için kullanılır.

`JSON` request'ini önceki örnekte kullanılan User nesnesine dönüştürmek için kullanılan bir metod yaratalım:

```java
@RequestMapping(value = "/isConverted", method = RequestMethod.POST)
@ResponseBody
public String isConvertedFromJson(@RequestBody User user) {
    return user.getUserName();
}
```

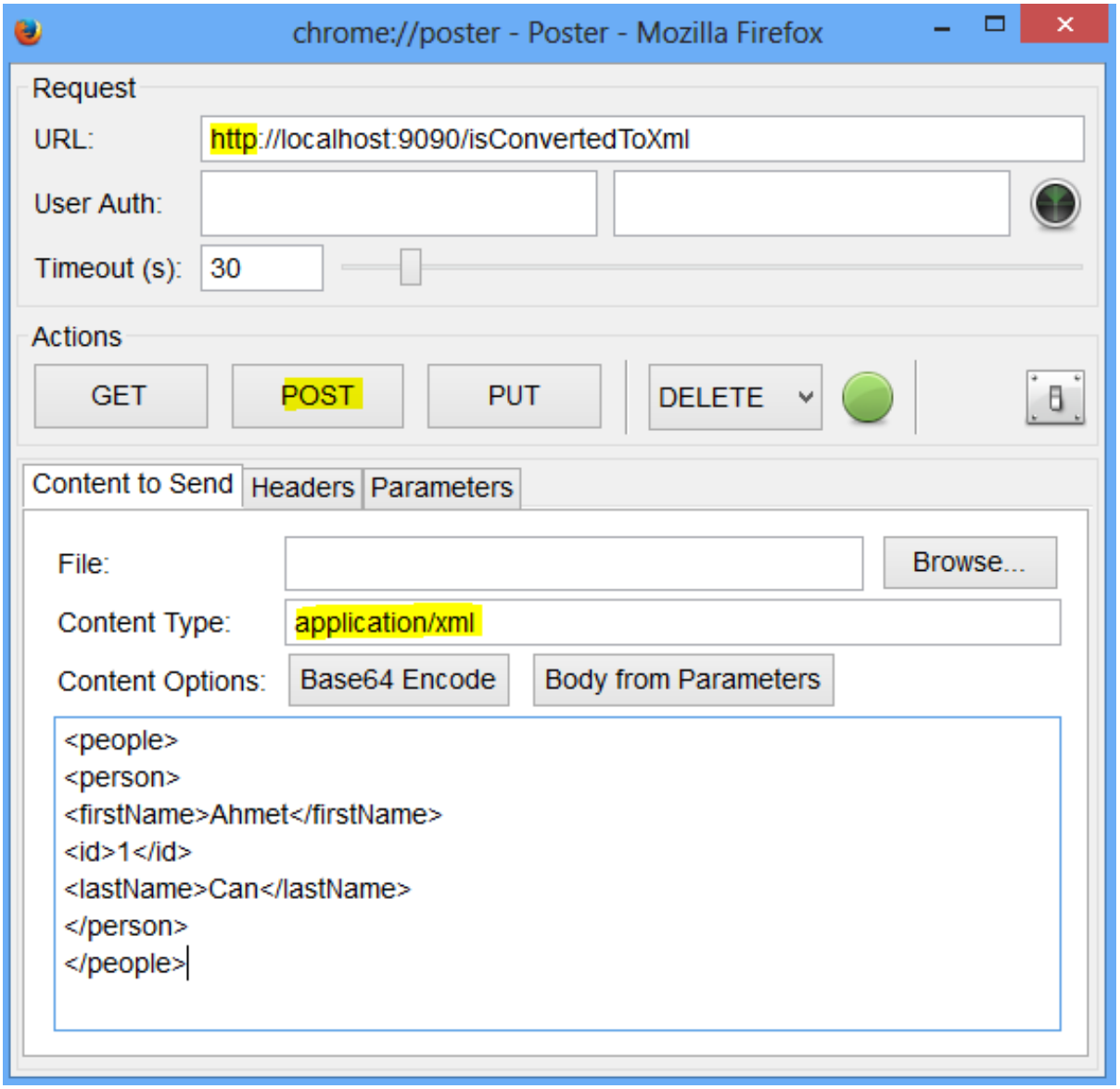
Bu örneği test edebilmek için Mozilla Firefox eklentisi olan **POSTER** (https://addons.mozilla.org/en-US/firefox/addon/poster/) plugini'ni kullanabiliriz:



XML request'ini önceki örnekte kullanılan People nesnesine dönüştürmek için kullanılan bir metod yaratalım:

```java
@RequestMapping(value = "/isConvertedToXml", method = RequestMethod.POST)
@ResponseBody
public String isConvertedFromXML(@RequestBody People people) {
    return people.getPeople().get(0).getFirstName();
}
```

Bu örneği test edebilmek için Mozilla Firefox eklentisi olan **POSTER** (https://addons.mozilla.org/en-US/firefox/addon/poster/) plugini'ni kullanabiliriz:



Not: **@RequestBody** annotasyonu setter metodlara ihtiyaç duyarken, **@ResponseBody** annotasyonu getter metodlara ihtiyaç duyar. Bundan dolayı **POJO** sınıfları oluştururken, getter ve setter metodlarının eklenmesi gereklidir.