

Generació dinàmica de pàgines web

Mercedes Castellón Fuentes, Miguel Angel Lozano Márquez, Sergi
Pérez Pérez

Desenvolupament web en entorn servidor

Índex

Introducció	5
Resultats d'aprenentatge	7
1 Introducció a Spring i Spring MVC	9
1.1 "Hola, Món" amb Maven	9
1.1.1 Preparació de l'entorn de desenvolupament	10
1.1.2 Actualització dels repositoris amb Maven	10
1.1.3 Creació d'un projecte amb Maven	11
1.2 'Hola, Món' web amb Maven	15
1.2.1 Creació d'un projecte nou	15
1.2.2 Estructura típica d'una aplicació web	17
1.3 "Hola, Món" amb Spring MVC	18
1.3.1 Creació del projecte "Hola, Món" a partir de Maven WebApp	18
1.3.2 Afegint dependències J2EE i Spring MVC	19
1.3.3 Arquitectura i procés bàsic d'una aplicació Spring MVC	20
1.3.4 Completant 'Hola, Món' amb l'arquitectura Spring MVC	22
1.3.5 Conclusions del nostre 'Hola, Món' Spring MVC	25
1.4 Estoc de medicaments, benvinguda	26
1.4.1 Creació i configuració inicial del projecte Estoc de medicaments	26
1.4.2 Pàgina de benvinguda	27
1.5 Què s'ha après?	29
2 Spring MVC, aplicació web	31
2.1 Estoc de medicaments, capa de domini	32
2.1.1 Creació del domini	33
2.1.2 Fent servir el domini	34
2.2 Estoc de medicaments, capa de persistència	37
2.2.1 Creació de la persistència	38
2.3 Estoc de medicaments, capa de servei	41
2.3.1 Adaptació del repositori	42
2.3.2 Creació del servei	43
2.3.3 Afegits a la capa de presentació	44
2.3.4 Què heu fet i què heu après amb la capa de servei?	46
2.4 Què s'ha après?	46
3 Spring MVC, aprofundint en els controladors	49
3.1 Estoc de medicaments, capa de servei a medicament	50
3.1.1 Creant el servei a medicament	50
3.1.2 Ordenant les crides des del controlador	51
3.2 Estoc de medicaments, medicaments per malaltia	52
3.2.1 Adaptant repositori i servei	53
3.2.2 Adaptant el controlador	53
3.3 "Estoc de medicaments", llista de medicaments segons filtre	55

3.3.1	Adaptant repositori i servei	55
3.3.2	Adaptant el controlador	57
3.4	"Estoc de medicaments", detall de medicament	58
3.4.1	Mostrant el detall	59
3.4.2	Arribant al detall des de la llista	61
3.5	Què s'ha après?	63
4	Spring MVC, altres aplicacions	65
4.1	Estoc de medicaments, formulari per afegir medicaments	66
4.1.1	Treballant les capes de domini, persistència i servei	67
4.1.2	Formulari a la capa de presentació	67
4.2	Estoc de medicaments, llista blanca al formulari	72
4.3	Estoc de medicaments, pàgina de 'login'	73
4.3.1	Configurant Spring Security	73
4.3.2	El controlador per al 'login'	76
4.3.3	La vistes	76
4.3.4	Provant d'entrar	78
4.4	Estoc de medicaments, internacionalització	79
4.4.1	Externalitzem els missatges	80
4.4.2	Configurant i implementant el canvi d'idioma	82
4.4.3	Provant a canviar d'idioma	82
4.5	Què s'ha après?	83

Introducció

El desenvolupament web en la part del servidor es pot fer amb diversos llenguatges i servidors d'aplicacions que els puguin executar. Cadascun d'aquests té els seus avantatges; per exemple, amb PHP la corba d'aprenentatge és ràpida, i amb JEE (Java Enterprise Edition) es poden desenvolupar aplicacions molt escalables i segures.

En aquesta unitat anirem una mica més enllà dels avantatges propis de JEE per afegir més productivitat en el desenvolupament i permetre un manteniment eficient i reusabilitat del codi molt gran. Això ho farem emprant el *framework* **Spring MVC** basat en Java Enterprise Edition, desenvolupant uns projectes preliminars seguits d'una veritable aplicació en aquest entorn.

Per introduir el *framework* explicarem les típiques aplicacions “Hola, Món” des de diversos punts de vista. En primer lloc, crearem projectes Maven amb l'estructura de qualsevol aplicació web fins a arribar a crear un “Hola, Món” amb l'estructura d'una aplicació web basada en Spring MVC.

Un cop feta la introducció, es començarà a desenvolupar tota una aplicació web que anomenarem “**Estoc de medicaments**”. L'anirem completant pas a pas explicant els continguts que hi afegim, és a dir, especificant els conceptes i la pràctica amb Spring MVC.

Farem una pàgina de benvinguda de l'aplicació que ens servirà per compondre l'estructura bàsica d'una aplicació web Spring MVC i descriure com una petició va des del navegador a un controlador frontal (*Dispatcher Servlet*) i aquest el dirigeix a un controlador específic.

Continuarem mostrant la llista de medicaments però desenvolupant totes les capes que Spring MVC recomana:

- presentació (*Presentation*)
- domini (*Domain*)
- serveis (*Services*)
- persistència (*Persistence*)

Aprofundirem en els controladors desenvolupant un moviment d'estoc en “Estoc de medicaments” i a partir de mostrar el detall. Veurem com podem passar paràmetres amb l'URL i amb el cos de la petició.

Finalment, veurem altres aplicacions del *framework*, com el tractament de formularis, l'autenticació i certs aspectes sobre la seguretat i la internacionalització.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Desenvolupa aplicacions web identificant i aplicant mecanismes per separar el codi de presentació de la lògica de negoci.

- Identifica els avantatges de separar la lògica de negoci dels aspectes de presentació de l'aplicació.
- Analitza tecnologies i mecanismes que permeten fer aquesta separació i les seves característiques principals.
- Utilitza objectes i controls en el servidor per generar l'aspecte visual de l'aplicació web en el client.
- Empra formularis generats de manera dinàmica per respondre als esdeveniments de l'aplicació web.
- Identifica i aplica els paràmetres relatius a la configuració de l'aplicació web.
- Escriu aplicacions web amb manteniment d'estat i separació de la lògica de negoci.
- Aplica els principis de la programació orientada a objectes.
- Aprova i documenta el codi.

1. Introducció a Spring i Spring MVC

Spring és un *framework* amb mòduls basat en Java Enterprise Edition. La principal característica del seu *core* és la utilització del patró de disseny inversió de control (IoC, per les sigles en anglès) i també la injecció de dependències (DI, per les sigles en anglès), un tipus d'IoC.

En concret, fent servir IoC i DI al nostre codi no crearem els objectes que necessitem de les llibreries del *framework*; simplement definint la configuració amb fitxers XML o amb anotacions al codi, Spring ens proporcionarà l'objecte adient.

Spring MVC és un mòdul d'Spring que ens ajuda a construir aplicacions sota el patró model vista controlador (MVC, per les sigles en anglès)

El model és el conjunt d'objectes que representen les dades de la nostra aplicació, la vista correspon a la manera de presentar les dades a l'usuari, i controlador manega les peticions fetes per l'usuari, interactuant amb la vista i amb el model.

A Spring MVC, qualsevol petició de l'usuari és recollida per un controlador central (`Dispatcher Servlet`) que determinarà, segons la configuració, el controlador específic de la nostra aplicació que haurà de recollir la petició. El nostre controlador haurà de construir la resposta amb la vista que correspon a la petició amb el model adient. El `Dispatcher Servlet` tornarà a prendre el control i retornarà la resposta al client.

I per fer independent els nostres projectes de l'IDE farem servir Maven com a eina que dóna estructura als projectes i que permet baixar les llibreries que fareu servir des de repositoris centrals a partir de la configuració de dependències.

Començareu creant dos projectes amb Maven sense dependències d'Spring MVC per conèixer l'estructura dels projectes i les dependències necessàries amb Java Enterprise Edition per continuar amb projectes Spring MVC, un "Hola, Món" amb Spring MVC per apropar-nos al *framework* i un projecte "Estoc de medicaments" que simplement mostra una pàgina de benvinguda.

1.1 "Hola, Món" amb Maven

Maven és una eina que us permet descarregar totes aquelles llibreries que necessita un servidor extern el qual conté un repositori amb un llistat molt ampli de llibreries i components; a més de poder descarregar llibreries, també us permet la generació de projectes de manera automàtica per a aplicacions de propòsit específic. Per exemple, una aplicació web, com seria el vostre cas.

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Tiene un modelo de configuración basado en un formato XML (archivo pom.xml). Actualmente es un proyecto perteneciente a la Apache Software Foundation.

Maven també permet transportar un projecte des d'un entorn de desenvolupament a un altre. Per exemple, si teniu un projecte Maven creat amb Eclipse el podreu importar sense problemes a NetBeans.

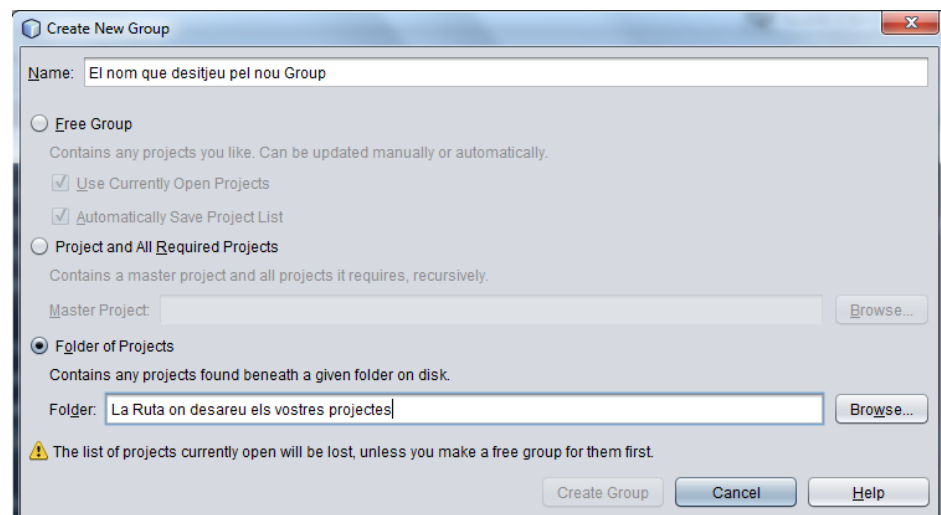
1.1.1 Preparació de l'entorn de desenvolupament

El primer que farem per preparar el vostre entorn serà configurar Netbeans per aïllar els vostres projectes dels projectes d'altres mòduls desenvolupats amb Netbeans, i d'aquesta manera aconseguireu un espai de treball molt més net.

Per tant, aneu al directori *NetBeansProjects* i allà creeu el directori *Spring_Projects*, que serà el que fareu servir per als vostres projectes amb Spring.

Un cop creat el directori, torneu a Netbeans i aneu a *File/Projects Group*, això us obrirà una finestra que us permetrà definir el directori que acabeu de crear com un nou espai de treball en blanc (vegeu la figura 1.1).

FIGURA 1.1. Creació d'un grup de projectes



1.1.2 Actualització dels repositoris amb Maven

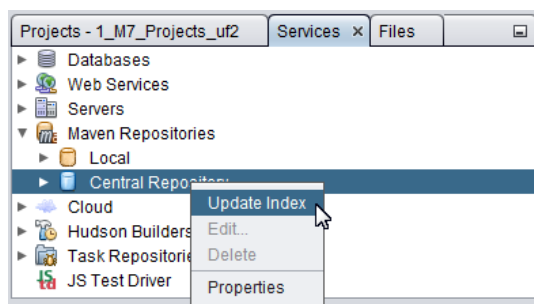
Un cop definit i seleccionat el nou espai de treball, el que fareu ara serà actualitzar els repositoris amb Maven.

Per això, en el vostre entorn de Netbeans aneu a la pestanya *Serveis* i allà desplegueu l'apartat *Maven Repositories*; com podeu observar, existeixen dos repositoris: un d'extern, o repositori central, i un local.

El que fareu serà establir una connexió amb el repositori central per tal d'obtenir una llista actualitzada de tots els components emmagatzemats al repositori central.

Per fer-ho, aneu a la pestanya *Services* de NetBeans, desplegueu l'element *Maven Repositories* i, situats al node *Central Repository*, prement amb el botó dret, podreu actualitzar la llista (opció *Update Index*), tal com es pot veure en la figura 1.2.

FIGURA 1.2. Actualització de l'índex de components

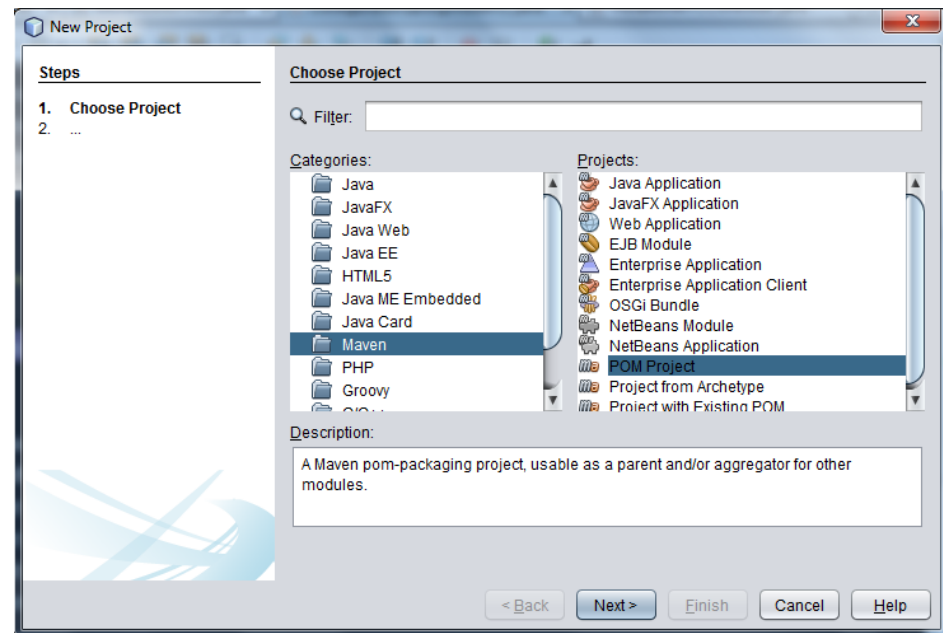


El procés d'actualització dels repositoris triga una bona estona; per tant, tingueu paciència.

1.1.3 Creació d'un projecte amb Maven

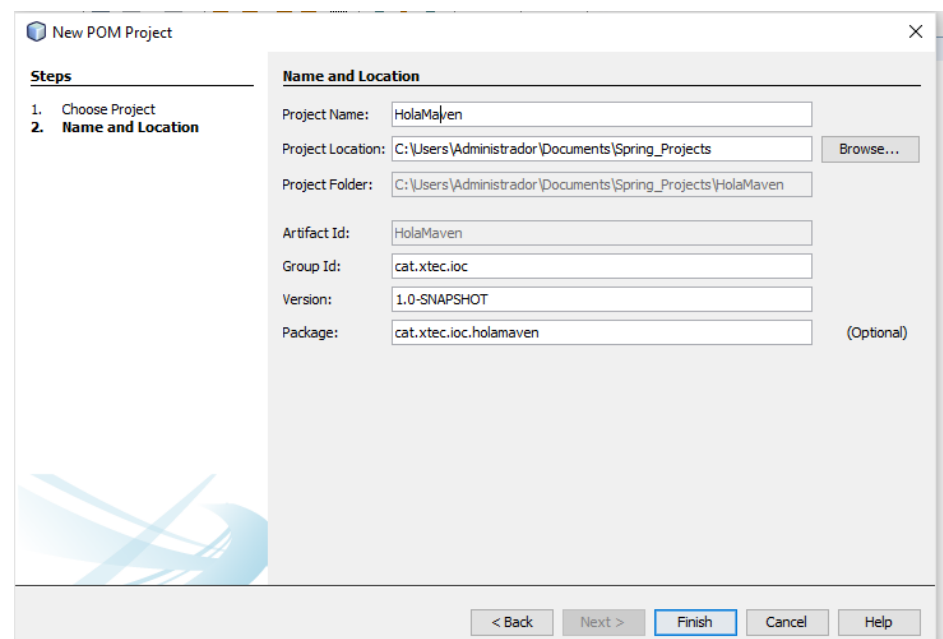
Un cop actualitzats els repositoris podeu procedir a la creació d'un projecte amb Maven; a partir d'aquest aprofundirem en els conceptes i l'estructura que en proporciona.

Per fer això aneu a *File/New Project*, en l'apartat de categories seleccioneu *Maven* i en l'apartat *Project* seleccioneu *POM Projects*. Ara només us resta definir el nom i la localització del vostre primer projecte amb Maven (vegeu la figura 1.3).

FIGURA 1.3. Nou projecte Maven

Aquí haureu d'especificar tres apartats: el nom del projecte, la localització del projecte i l'identificador del grup (vegeu la figura 1.4).

- nom del projecte: HolaMaven
- localització del projecte: Spring_Projects
- identificador del grup: cat.xtec.ioc

FIGURA 1.4. Propietats del projecte

Si tot ha anat bé, tant al directori *Spring_Projects* com a Netbeans podreu veure que s'ha creat un nou projecte.

Definició de les dependències

Les dependències no són altra cosa que les llibreries o els components que necessita la nostra aplicació per funcionar o portar a terme una tasca determinada. Un exemple de dependència seria la llibreria JDBC per a MySQL, que ens permet obrir i tancar connexions contra una base de dades MySQL i fer diferents operacions contra aquesta.

Per desenvolupar una aplicació web necessitem descarregar tres dependències des dels repositoris de Maven, que són:

- l'entorn de treball Spring
- la llibreria jstl
- la llibreria servlet-api

Maven fa servir un fitxer de configuració XML anomenat `pom.xml` on podeu indicar les dependències de la vostra aplicació web; aquest fitxer el podeu localitzar a la carpeta *Projectes Files* del vostre nou projecte. Però com podeu veure si l'obriu, no fa cap esment de Spring ni a la resta de llibreries que comentàvem.

L'edició del fitxer `pom.xml` per incloure aquestes tres llibreries la podem portar a terme de dues maneres: una seria fent-la directament contra el fitxer `pom.xml` i l'altra seria utilitzant l'assistent que ens ofereix NetBeans per tal de fer aquesta edició contra el fitxer `pom.xml`.

En el vostre cas, i per començar, fareu servir l'assistent, ja que us ajudarà a comprendre d'una manera més senzilla què esteu fent.

Així doncs, amb el fitxer `pom.xml` obert, aneu a la carpeta *Dependències* del vostre projecte i feu clic amb el botó dret del ratolí. Això obrirà un desplegable amb l'opció *Add Dependencie...*; feu clic sobre aquesta opció i s'obrirà una finestra nova.

En aquesta finestra apareixen diferents camps per omplir, però a nosaltres només ens interessen tres, que són:

- *Group Id*: es correspon amb l'identificador que fa servir Maven per a un conjunt de components desenvolupats per un determinat projecte o empresa. Aquest identificador pren l'aspecte de l'espai de nom del projecte, com per exemple *org.springframework*.
- *Artifact Id*: es correspon amb un component determinat desenvolupat per un projecte o empresa; en general, és el nom del component o llibreria.
- *Version*: representa la versió del component que volem fer servir.

Ompliu els tres camps que hem comentat amb els següents valors, respectivament:

- `org.springframework`

- spring-webmvc
- 4.0.3.RELEASE

Afegiu la dependència i observeu què ha passat amb el fitxer pom.xml i amb la carpeta *Dependències*.

Com podeu veure, el fitxer pom.xml s'ha modificat amb noves etiquetes i valors, i la carpeta *Dependències* incorpora ara noves llibreries que es corresponen amb les llibreries que ens permeten desenvolupar aplicacions web amb l'entorn de treball Spring.

Si haguéssiu tingut un error d'escriptura dels valors indicats per a la dependència, Maven seria incapaç de descarregar-la i ho indicaria amb un petit triangle groc d'avertència sobre la icona del component a la carpeta *Dependències*.

Un cop descarregat el primer component, descarregueu els altres dos amb l'assistent. Les dades que necessiteu són les següents:

- Per a la llibreria jstl
 - javax.servlet
 - jstl
 - 1.2
- Per a la llibreria servlet-api
 - javax.servlet
 - javax.servlet-api
 - 3.1.0

El fitxer pom.xml del projecte quedarà de la següent manera:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5                             http://maven.apache.org/xsd/maven-4.0.0.xsd"
6         >
7     <modelVersion>4.0.0</modelVersion>
8     <groupId>cat.xtec.ioc</groupId>
9     <artifactId>A0010_Spring_Introduccio</artifactId>
10    <version>1.0-SNAPSHOT</version>
11    <packaging>pom</packaging>
12    <dependencies>
13        <dependency>
14            <groupId>org.springframework</groupId>
15            <artifactId>spring-webmvc</artifactId>
16            <version>4.0.3.RELEASE</version>
17        </dependency>
18        <dependency>
19            <groupId>javax.servlet</groupId>
20            <artifactId>jstl</artifactId>
21            <version>1.2</version>
22        </dependency>
23        <dependency>
24            <groupId>javax.servlet</groupId>
```

```
24     <artifactId>javax.servlet-api</artifactId>
25     <version>3.1.0</version>
26   </dependency>
27 </dependencies>
28 <properties>
29   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
30 </properties>
31 </project>
```

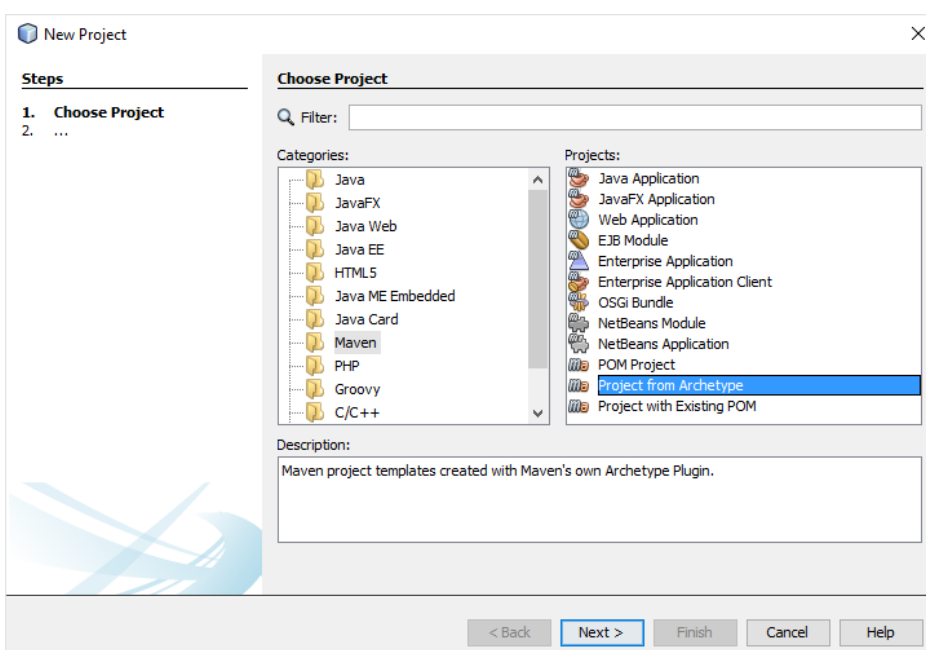
1.2 'Hola, Món' web amb Maven

La creació d'aquest segon projecte la farem mitjançant un arquetipus, que és la definició d'una estructura de directoris i fitxers específics per a un projecte genèric, i el que fa és permetre la creació d'un projecte nou seguint una convenció determinada; en el vostre cas, es crearà una estructura que segueix la convenció establerta per a la creació d'aplicacions web amb Spring.

1.2.1 Creació d'un projecte nou

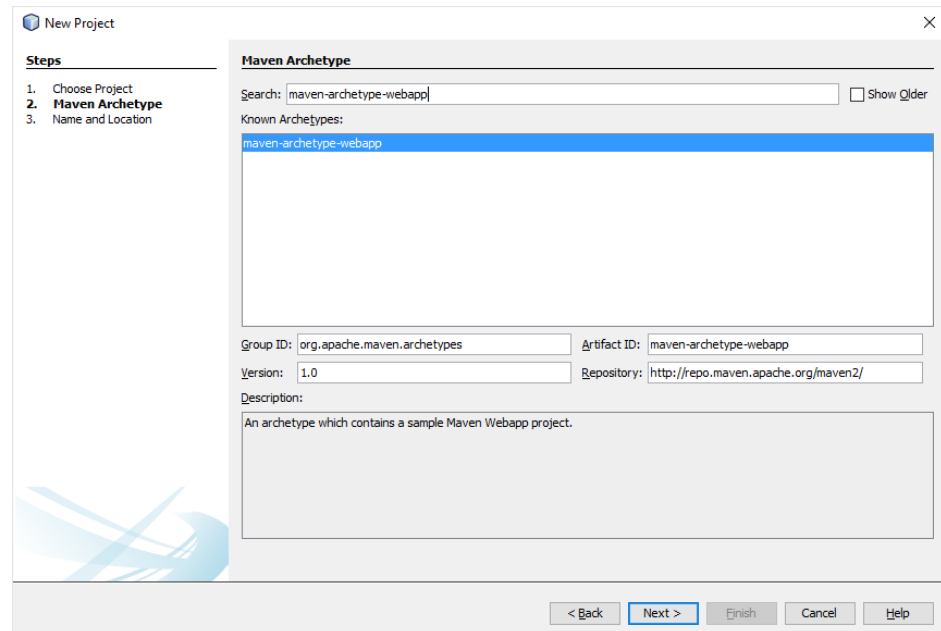
Per crear aquest nou projecte web aprofitarem la carpeta *Modules* del projecte Maven creat anteriorment. Si feu clic sobre la carpeta amb el botó dret de ratolí s'obrirà un menú contextual, on podreu seleccionar l'opció *Create New Module*. Això obrirà un nou assistent per a la creació de projectes. Aquí seleccionarem la categoria *Maven*, i dintre del tipus de projectes disponibles seleccionarem *Project from Archetype* (vegeu la figura 1.5).

FIGURA 1.5. Nou projecte Maven from Archetype



Un cop seleccionat el nou tipus d'aplicació a desenvolupar, cercareu l'arquetipus desitjat; en el vostre cas heu de cercar el següent arquetipus: `maven-archetype-webapp`. Un cop localitzat l'arquetipus, el seleccionareu i pulsareu el següent en l'assistent (vegeu la figura 1.6).

FIGURA 1.6. Propietats de l'arquetipus

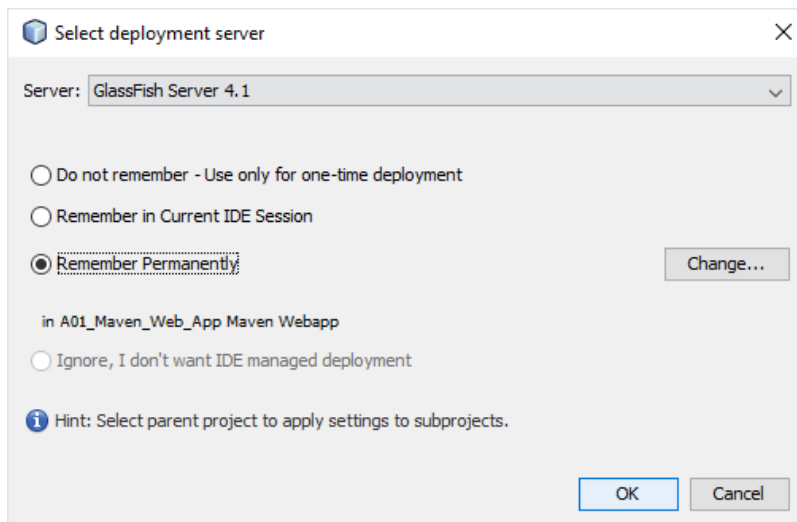


Això obrirà una nova finestra en l'assistent, on donareu el següent nom al projecte: `A01_Maven_Web_App`, i verificareu que tant la localització del projecte (`Spring_Projects`) com el Group Id (`cat.xtec.ioc`) són els correctes, i un cop fet això finalitzareu l'assistent.

En tancar l'assistent es generarà un nou mòdul, així com un nou projecte. Si el nom no es correspon amb el que li havíeu indicat prèviament podeu reanomenar-lo amb el botó dret de ratolí obrint el menú contextual i seleccionant l'opció *Reanomenar*. Aquest punt l'heu de fer des del projecte generat, no des del mòdul. Finalment, amb el mateix menú contextual teniu l'opció d'executar el nou projecte web clicant sobre l'apartat *Run*. En aquest punt s'obrirà un nou assistent on haureu d'indicar qui serà el servidor de desplegament de l'aplicació; seleccioneu *GlassFish Server 4.1* (o la versió més recent) i l'opció *Recorda de manera permanent* (vegeu la figura 1.7).

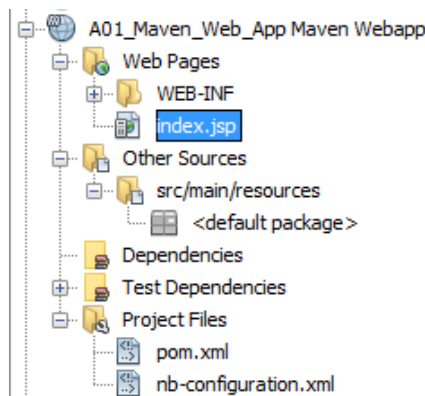
Després d'això, Netbeans procedirà al desplegament de la nostra aplicació, engegarà el servidor d'aplicacions GlassFish, executarà l'aplicació i podreu veure el resultat a través del vostre navegador web.

Si el desplegament s'ha fet correctament podeu anar a la pestanya *Services* de NetBeans i veure la vostra aplicació web com a node de *Servers/Glassfish Server xxx/Applications*, on "xxx" és la versió del servidor que heu fet servir.

FIGURA 1.7. Selecció del servidor d'aplicacions

1.2.2 Estructura típica d'una aplicació web

Si us situeu sobre el projecte `A01_Maven_Web_App`, la vostra primera aplicació té la següent estructura des de la pestanya projectes: *Web Pages*, *Other Sources*, *Dependencies*, *Test Dependencies* i *Project Files* (vegeu la figura 1.8).

FIGURA 1.8. Estructura d'un projecte Maven WebApp

La carpeta *Project Files* conté el fitxer `pom.xml`, que és on es declaren les dependències del projecte. Com podeu observar, el fitxer `pom.xml` del mòdul (projecte `A01_Maven_Web_App`) és diferent del fitxer `pom.xml` del projecte arrel (projecte `A00_HolaMaven`); tot i això, si obriu la carpeta *Dependencies* del projecte `A01_Maven_Web_App` podreu observar que fa referència a les dependències declarades en el projecte arrel.

La carpeta *Test Dependencies* inclourà les llibreries necessàries per fer les proves de la nostra aplicació. *Other Sources* és una carpeta que no farem servir per ara, i finalment *Web Pages* contindrà tot allò vinculat a les vistes web de l'aplicació. Dintre de *Web Pages* hi ha la carpeta *WEB-INF*, que serà l'arrel de totes les vistes generades.

Generalment, *WEB-INF*, a banda dels *scripts* jsp, contindrà el descriptor de desplegament *web.xml* i els fitxers de configuració de Spring que veurem més endavant.

1.3 "Hola, Món" amb Spring MVC

Fins ara heu après a crear projectes amb Maven, gestionar les dependències i crear una aplicació web molt senzilla però sense fer servir Spring.

Ara farem la nostra primera aplicació amb el *framework* Spring MVC amb l'objectiu de mostrar com es configura tot l'entorn d'un projecte desenvolupat amb aquest *framework*.

L'aplicació serà un "Hola, Món" per mostrar simplement aquest text en el navegador, però amb l'estructura d'un projecte Maven WebApp i els components de Spring MVC.

El projecte sencer de l'aplicació es pot baixar des del següent .

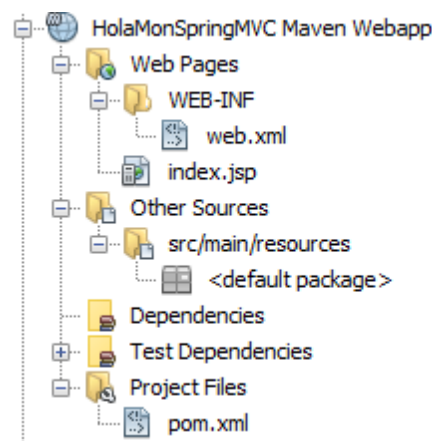
És convenient fer servir el projecte anterior per consultar, però és millor anar construint el vostre propi projecte amb les indicacions que es van donant. No obstant això, si voleu executar el projecte sencer segurament us obligarà a fer *clean and build* abans.

1.3.1 Creació del projecte "Hola, Món" a partir de Maven WebApp

Creeu un nou projecte *Maven/Project from Archetype* amb l'arquetipus *maven-archetype-webapp* i anomeu-lo *HolaMonSpringMVC*.

Recordem l'estructura del projecte creat fins ara: només és un projecte Maven WebApp a tots els efectes (vegeu la figura 1.9).

FIGURA 1.9. Estructura Maven d'una aplicació Spring MVC



Aquesta estructura correspon a qualsevol tipus d'aplicació, i per això, des de la pestanya *Files* de NetBeans o des del mateix explorador de fitxers del sistema operatiu creeu la carpeta *Java* dins de la carpeta *src/main*. Ara, si tornem a la pestanya *Projects* de NetBeans, veurem que ha aparegut la carpeta *Source Packages* on crear els nostres paquets.

Abans de continuar, canvieu el fitxer de configuració *web.xml* a una versió més moderna de JAVA EE.

```
1 <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
2     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
4         http://java.sun.com/xml/ns/javaee/web-app_3_0.
5         xsd">
6     <display-name>HolaMonSpringMVC</display-name>
</web-app>
```

Ara ja podeu executar l'aplicació seleccionant el servidor Glassfish i fent que recordi aquesta decisió permanentment (vegeu la figura 1.10).

FIGURA 1.10. Sortida de l'aplicació "Hola, Món" Spring MVC

Hello World!

1.3.2 Afegint dependències J2EE i Spring MVC

Anem a afegir les dependències que faran que el servidor d'aplicacions Glassfish, en executar la nostra aplicació, pugui crear els objectes necessaris de J2EE i Spring MVC i així aprofitar la funcionalitat que aquests entorns ens ofereixen.

A la secció *dependències* de *pom.xml* heu d'afegir-hi les següents:

```
1 <dependency>
2     <groupId>org.springframework</groupId>
3     <artifactId>spring-webmvc</artifactId>
4     <version>4.0.3.RELEASE</version>
5 </dependency>
6 <dependency>
7     <groupId>javax.servlet</groupId>
8     <artifactId>jstl</artifactId>
9     <version>1.2</version>
10 </dependency>
11 <dependency>
12     <groupId>javax.servlet</groupId>
13     <artifactId>javax.servlet-api</artifactId>
14     <version>3.1.0</version>
15 </dependency>
16 <dependency>
17     <groupId>javax</groupId>
18     <artifactId>javaee-web-api</artifactId>
19     <version>7.0</version>
20 </dependency>
```

A més, aprofitareu per fer servir la codificació UTF8 en tot el projecte afegint la propietat que defineix aquesta característica en el mateix fitxer pom.xml:

```

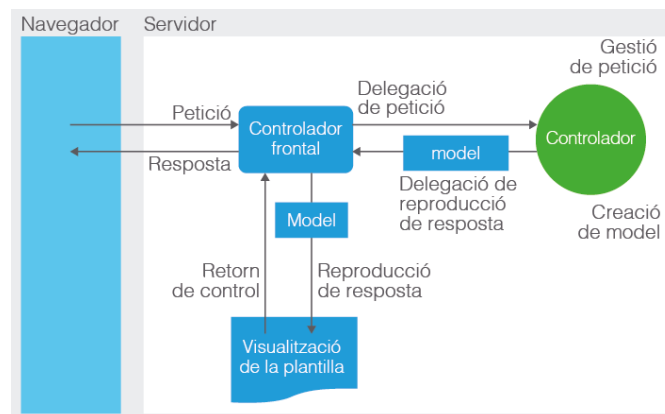
1 <properties>
2   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
3 </properties>

```

1.3.3 Arquitectura i procés bàsic d'una aplicació Spring MVC

Des que un navegador fa una petició al servidor d'aplicacions, per exemple Glassfish, fins que el servidor construeix la resposta i retorna al navegador, el flux de l'aplicació és el que es mostra en la figura 1.11.

FIGURA 1.11. Arquitectura i procés d'una petició amb Spring MVC



A continuació es descriu la seqüència del procés petició-resposta.

1. El navegador demana (petició) un recurs a l'aplicació web.
2. El Front Controller o Dispatcher Servlet, implementat com un *servlet*, intercepta la petició i la delega al controlador adient, que ha de gestionar (*handler*) aquesta petició. Com veurem més endavant, la determinació del controlador adient es fa mitjançant la configuració dels *Handler Mappings*.
3. El controlador processa la petició i retorna al Front Controller el model i la vista. Aquest retorn es fa amb un objecte del tipus *ModelAndView*.
4. El Front Controller resol la vista actual (per exemple, un jsp) consultant l'objecte *ViewResolver*.
5. La vista seleccionada és retornada com a resposta al navegador que va fer la petició.

Front Controller

Front Controller és un patró de disseny que proposa la centralització de la gestió de les peticions i respostes. Cerqueu "front controller java enterprise edition" i trobareu documentació d'Oracle sobre aquest tema.

Web Application Context

Desplegar una aplicació Spring al servidor d'aplicacions, per exemple a Glassfish, és com si l'aplicació s'estigués executant en segon pla dins del servidor.

Els objectes de la nostra aplicació desplegada resideixen en **contenidors**.

Per a tota l'aplicació hi ha un contenidor arrel anomenat *Application Context* que conté els objectes definits al fitxer de configuració anomenat `applicationContext.xml`.

El nom del fitxer de configuració per a *Application Context* es pot canviar a `web.xml`, com mostra l'exemple següent.

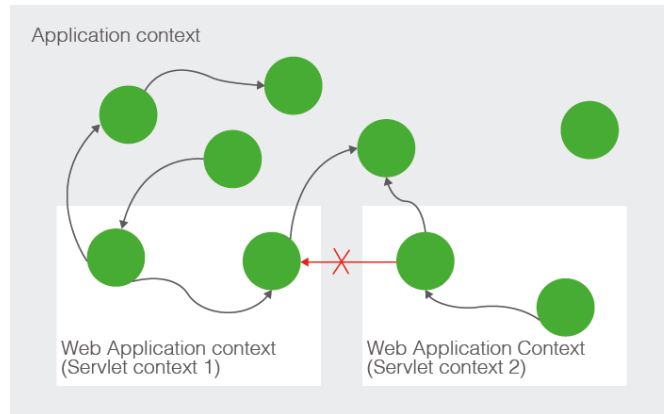
```
1 <context-param>
2   <param-name>contextConfigLocation</param-name>
3   <param-value>/WEB-INF/rootApplicationContext.xml</param-value>
4 </context-param>
```

Per a cada Dispatcher Servlet es crearà un contenidor específic anomenat *Web Application Context*. Per defecte, el seu fitxer de configuració s'anomena `xxx-servlet.xml`, on "xxx" és el nom del Dispatcher Servlet.

El nom del fitxer per a *Web Application Context* es configura al mateix `web.xml`. L'exemple següent mostra com es configura un Dispatcher Servlet amb nom Dispatcher Servlet i amb fitxer de configuració `/WEB-INF/spring/DispatcherServlet-servlet.xml`.

```
1 <servlet>
2   <servlet-name>DispatcherServlet</servlet-name>
3   <servlet-class>org.springframework.web.servlet.DispatcherServlet</
4     servlet-class>
5   <init-param>
6     <param-name>contextConfigLocation</param-name>
7     <param-value>/WEB-INF/spring/DispatcherServlet-servlet.xml</param-
8       value>
9   </init-param>
10  <load-on-startup>1</load-on-startup>
11 </servlet>
```

Com es veurà més endavant, els fitxers de configuració de context, tant el de tota l'aplicació com els específics de cada Dispatcher Servlet, defineixen els objectes que es crearan al contenidor. La diferència és que els objectes a nivell d'aplicació es poden fer servir des de qualsevol contenidor, i els de nivell de Dispatcher Servlet, únicament des d'objectes del mateix contenidor. A més, els del contenidor a nivell d'aplicació poden fer servir qualsevol objecte de qualsevol contenidor específic, tal com es vol reflectir a la figura [1.12](#).

FIGURA 1.12. Visibilitat d'objectes segons els nivells

En aquests fitxers de configuració es defineixen quins objectes formaran part del contenidor, tant si són propis de l'aplicació com si són proporcionats per Spring o Java Enterprise Edition. Aquests objectes s'anomenen *Spring-managed beans* o simplement *beans*, i els farem servir al nostre codi declarant-los mitjançant anotacions, és a dir, no els instanciaré, perquè farem servir els objectes del contenidor. Per això es diu que l'objecte s'**injeta** quan es necessita i podem fer-lo servir (**injecció de dependències** o **inversió de control**)

Quan es dissenya una aplicació és important tenir en compte que els *beans* que defineixen la lògica de negoci, la interacció amb la persistència i altres interaccions s'han de compartir entre tots els *servlets*, i per això els definirem a nivell d'*Application Context* (el contenidor general) En canvi, els controladors que manegen peticions, els View Resolvers i altres com alguns que gestionen missatges, els ubicarem a nivell de Web Application Context (els específics de cada Dispatcher Servlet).

Injecció de dependències

La injecció de dependències o inversió de control aconsegueix codi més desacoblant, ens facilita els tests i, a més, ens permetrà canviar bocins de codi de manera més fiable i ràpida.

1.3.4 Completant 'Hola, Món' amb l'arquitectura Spring MVC

Un cop sabeu com ha de ser el flux d'una aplicació Spring MVC i els components que ha de tenir, desenvolupau les classes i la configuració que falten a la nostra aplicació "Hola, Món".

En el vostre cas, deixareu que Spring agafi la configuració per defecte a nivell d'*Application Context*. Per això no creareu cap fitxer de configuració de l'*Application Context* (applicationContext.xml).

El que sí que hem de crear és la configuració del Web Application Context. Com només tindreu un Dispatcher Servlet amb el nom Dispatcher Servlet, podeu crear una nova carpeta de nom *spring* dins de *WEB-INF* i dins el fitxer DispatcherServlet-servlet.xml. El contingut és el que es mostra a continuació.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:context="http://www.springframework.org/schema/context"
5     xmlns:mvc="http://www.springframework.org/schema/mvc"

```

```

6      xsi:schemaLocation="http://www.springframework.org/schema/beans
7                          http://www.springframework.org/schema/beans/spring-
8                          beans.xsd
9                          http://www.springframework.org/schema/context
10                         http://www.springframework.org/schema/context/spring-
11                         -context-4.0.xsd
12                         http://www.springframework.org/schema/mvc
13                         http://www.springframework.org/schema/mvc/spring-mvc
14                         -4.0.xsd">
15
16  <mvc:annotation-driven />
17  <context:component-scan base-package="cat.xtec.ioc" />
18
19  <bean class="org.springframework.web.servlet.view.
20      InternalResourceViewResolver">
21      <property name="prefix" value="/WEB-INF/views/" />
22      <property name="suffix" value=".jsp" />
23  </bean>
24
25 </beans>

```

Amb `<mvc:annotation-driven/>` li diem a Spring MVC que farem servir els *beans* `DefaultAnnotationHandlerMapping`, `AnnotationMethodHandlerAdapter` i `ExceptionHandlerExceptionResolver`. Aquests *beans* són necessaris per enviar (*dispatch*) les peticions als controladors que les maneguen.

Com s'ha dit a la part d'arquitectura i procés de Spring MVC, el `DispatcherServlet` ha d'identificar quin és el controlador que manegará la petició. Per fer això cercarà tots els objectes amb l'anotació `@Controller` dels paquets indicats en la propietat `base-package` de l'etiqueta `context:component-scan`.

La darrera etiqueta *bean* indica a Spring la creació d'un objecte bean a partir de la classe `InternalResourceViewResolver`. Un View Resolver ajuda el `DispatcherServlet` a construir la resposta a partir de determinada vista. Spring MVC proporciona diverses implementacions de View Resolvers, i `InternalResourceViewResolver` és una d'aquestes.

Per definir el nostre Front Controller afegiu el següent contingut al fitxer de configuració `web.xml`.

```

1 <servlet>
2     <servlet-name>DispatcherServlet</servlet-name>
3     <servlet-class>org.springframework.web.servlet.DispatcherServlet</
4     servlet-class>
5     <init-param>
6         <param-name>contextConfigLocation</param-name>
7         <param-value>/WEB-INF/spring/DispatcherServlet-servlet.xml</param-
8         value>
9     </init-param>
10    <load-on-startup>1</load-on-startup>
11 </servlet>
12
13 <servlet-mapping>
14     <servlet-name>DispatcherServlet</servlet-name>
15     <url-pattern>/</url-pattern>
16 </servlet-mapping>

```

Fixeu-vos que es fa referència al fitxer de configuració `DispatcherServlet-servlet.xml` que hem creat. Podem canviar el nom del fitxer de configuració, però hauríeu de canviar-lo també aquí.

Continuant la compleció de la vostra aplicació seguint l'arquitectura i el model bàsic de Spring MVC, només resta crear el controlador que manejarà les peticions.

El *Controlador* és el responsable de processar les peticions, construir el model apropiat i passar-lo com a resposta a la vista que calgui.

Creeu el paquet *cat.xtec.ioc* i a dins la classe Java *HolaController* amb el contingut que es mostra a continuació.

```
1 package cat.xtec.ioc;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.ModelMap;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RequestMethod;
11 import org.springframework.web.servlet.ModelAndView;
12
13 @Controller
14 @RequestMapping("/")
15 public class HolaController {
16
17     @RequestMapping(method = RequestMethod.GET)
18     public ModelAndView handleRequest(HttpServletRequest request,
19                                     HttpServletResponse response)
20         throws ServletException, IOException {
21         ModelAndView modelview = new ModelAndView("resposta");
22         modelview.getModelMap().addAttribute("salutacio", "Hola a tothom");
23         return modelview;
24     }
25
26     @RequestMapping(value = "/bondia", method = RequestMethod.GET)
27     public ModelAndView handleRequestDia(HttpServletRequest request,
28                                         HttpServletResponse response)
29         throws ServletException, IOException {
30         ModelAndView modelview = new ModelAndView("resposta");
31         modelview.getModelMap().addAttribute("salutacio", "Bon dia a tothom");
32         return modelview;
33     }
34
35     @RequestMapping(value = "/bonanit", method = RequestMethod.GET)
36     public ModelAndView handleRequestNit(HttpServletRequest request,
37                                         HttpServletResponse response)
38         throws ServletException, IOException {
39         ModelAndView modelview = new ModelAndView("resposta");
40         modelview.getModelMap().addAttribute("salutacio", "Bona nit a tothom");
41         return modelview;
42     }
43 }
```

L'anotació *@Controller* és molt bàsica, però suficient en aquesta implementació.

L'anotació *@RequestMapping* serveix per determinar l'origen de la petició i així poder decidir quin mètode manejarà el retorn. Com podeu veure, es pot fer a nivell de la classe o a nivell de mètode. Els paràmetres d'aquesta anotació els podeu consultar a la documentació de referència de Spring; en aquest cas hem fet servir el tipus de petició i l'URL.

Cada mètode encarregat (*handler*) de construir el retorn crea un nou objecte *ModelAndView* amb el nom de la vista que ha de retornar. Aquest

nom, en combinació amb la configuració feta a `DispatcherServlet-servlet.xml` (`InternalResourceViewResolver`), serveix perquè Spring MVC determini exactament la vista (`WEB-INF/views/resposta.jsp`).

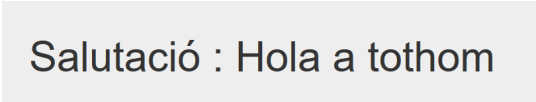
Als mètodes esmentats, abans de retornar la vista, s'afegeix un atribut al model (`ModelMap`) que en el nostre exemple serveix per provar diferents orígens a les peticions i veure diferents respostes.

Finalment, creeu la carpeta `views` dins de `WEB-INF`, i a dins el fitxer `resposta.jsp` amb el contingut que es mostra a continuació.

```
1 <%@ page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html lang="ca">
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/
7       bootstrap.min.css">
8     <title>Salutació</title>
9   </head>
10  <body>
11    <section>
12      <div class="jumbotron">
13        <div class="container">
14          <h1>Salutació : ${salutacio} </h1>
15        </div>
16      </div>
17    </section>
18  </body>
19 </html>
```

En aquest cas, quan es mostri la vista al navegador, el valor de la variable `salutacio` canviarà en funció de l'URL de petició. Executeu l'aplicació sense més, i us ha de donar la salutació estàndard (vegeu la figura 1.13).

FIGURA 1.13. Sortida de l'aplicació "Hola, Món" amb Spring MVC



Salutació : Hola a tothom

Si al navegador canvieu l'URL afegint-hi `/bondia` o `/bonanit`, les salutacions canviaran. Proveu-ho.

1.3.5 Conclusions del nostre 'Hola, Món' Spring MVC

L'aplicació simple Hola, Món està creada a partir d'una petició des d'un URL al navegador mostra diferents missatges. L'estructura està feta amb Maven WebApp, però després hi ha les dependències necessàries per a J2EE i Spring MVC al fitxer `pom.xml`.

Una aplicació Spring MVC té una arquitectura de classes i un procés determinat. Tota petició és rebuda per un Front Controller (`Dispatcher Servlet`) confi-

gurat a `web.xml`. Aquest la dirigeix cap al controlador que realment la manejarà (*handler*), però en realitat és el controlador (*HolaController*, en el nostre exemple) qui mitjançant les anotacions agafarà les peticions, construirà un `ModelAndView` i el retornarà al Front Controller per tal que aquest el torni al client (el navegador).

Per tal que tot funcioni correctament hem configurat la resolució dinàmica de les anotacions i la resolució de la vista efectiva mitjançant el fitxer `DispatcherServlet-servlet.xml`.

1.4 Estoc de medicaments, benvinguda

Desenvoluparem la pàgina d'inici (benvinguda) d'una aplicació per gestionar **estocs de medicaments** que anomenarem “**stmedioc**”. Crearem el projecte amb l'arquitectura d'una aplicació web basada en Spring MVC i amb la configuració basada en XML.

Configuració dels 'beans'

La configuració dels *beans* pot ser definida via fitxers de configuració XML, com `applicationContext.xml` i també via classes de configuració Java (`JavaConfig`).

1.4.1 Creació i configuració inicial del projecte Estoc de medicaments

És convenient fer servir el projecte anterior per consultar, però és millor anar construint el vostre propi projecte amb els indicacions que es van donant. No obstant això, si voleu executar el projecte sencer segurament us obliga a fer *clean and build* abans.

Creeu un nou projecte *Maven/Project from Archetype* amb l'arquetip `maven-archetype-webapp` i anomeu-lo “`stmedioc101`”.

Des de la pestanya *Files* de NetBeans o des del mateix explorador de fitxers del sistema operatiu, creeu la carpeta *Java* dins de la carpeta *src/main*. És important fer aquest pas, perquè veureu Source Packages a la pestanya de *Projectes*.

Canvieu el fitxer de configuració `web.xml`.

```
1 <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
2       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
4                           http://java.sun.com/xml/ns/javaee/web-app_3_0.
5                           xsd">
6   <display-name>Estoc de Medicaments</display-name>
</web-app>
```

A la secció *Dependències* de `pom.xml` heu d'afegir les següents dependències:

```
1 <dependency>
2   <groupId>org.springframework</groupId>
3   <artifactId>spring-webmvc</artifactId>
4   <version>4.0.3.RELEASE</version>
```

El codi del projecte “`stmedioc`” en l'estat d'aquest apartat es pot descarregar des de l'enllaç que trobareu als annexos de la unitat.

```

5      </dependency>
6      <dependency>
7          <groupId>javax.servlet</groupId>
8          <artifactId>jstl</artifactId>
9          <version>1.2</version>
10     </dependency>
11     <dependency>
12         <groupId>javax.servlet</groupId>
13         <artifactId>javax.servlet-api</artifactId>
14         <version>3.1.0</version>
15     </dependency>
16     <dependency>
17         <groupId>javax</groupId>
18         <artifactId>javaee-web-api</artifactId>
19         <version>7.0</version>
20     </dependency>

```

Afegiu la propietat que defineix la utilització de la codificació UTF-8 en el mateix fitxer pom.xml.

```

1 <properties>
2     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
3 </properties>

```

Executeu l'aplicació seleccionant el servidor **Glassfish** i fent que recordi aquesta decisió permanentment.

1.4.2 Pàgina de benvinguda

Creareu una pàgina de benvinguda per a la vostra aplicació consistent a mostrar un bàner però amb el procés de Spring MVC.

Per fer això haureu de crear la pàgina jsp, configurar el Dispatcher Servlet i crear el controlador per manegar la petició de benvinguda.

Creeu el fitxer DispatcherServlet-servlet.xml en una nova carpeta de nom *spring* dins de *WEB-INF*. El contingut és el que es mostra a continuació.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xmlns:mvc="http://www.springframework.org/schema/mvc"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7                           http://www.springframework.org/schema/beans/spring-
8                               beans.xsd
9                           http://www.springframework.org/schema/context
10                              http://www.springframework.org/schema/context/spring-
11                                  context-4.0.xsd
12                              http://www.springframework.org/schema/mvc
13                                  http://www.springframework.org/schema/mvc/spring-mvc-
14                                      4.0.xsd">
15
16     <mvc:annotation-driven />
17     <context:component-scan base-package="cat.xtec.ioc.controller" />
18
19     <bean class="org.springframework.web.servlet.view.
20         InternalResourceViewResolver">
21         <property name="prefix" value="/WEB-INF/views/" />

```

```

18     <property name="suffix" value=".jsp" />
19   </bean>
20 </beans>

```

En aquesta configuració es podrien tenir més paquets en *base-package* simplement separant-los amb comes.

Definim el Dispatcher Servlet (Front Controller) afegint el següent contingut al fitxer de configuració web.xml:

```

1 <servlet>
2   <servlet-name>DispatcherServlet</servlet-name>
3   <servlet-class>org.springframework.web.servlet.DispatcherServlet</
   servlet-class>
4   <init-param>
5     <param-name>contextConfigLocation</param-name>
6     <param-value>/WEB-INF/spring/DispatcherServlet-servlet.xml</param-
   value>
7   </init-param>
8   <load-on-startup>1</load-on-startup>
9 </servlet>
10 <servlet-mapping>
11   <servlet-name>DispatcherServlet</servlet-name>
12   <url-pattern>/</url-pattern>
13 </servlet-mapping>

```

Pel que fa al controlador que gestionarà les peticions, creeu el paquet `cat.xtec.ioc.controller` i a dins la classe Java `HomeController` amb el contingut que es mostra a continuació:

```

1 package cat.xtec.ioc.controller;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import org.springframework.web.bind.annotation.RequestMethod;
10 import org.springframework.web.servlet.ModelAndView;
11
12 @Controller
13 public class HomeController {
14
15     @RequestMapping(value = "/", method = RequestMethod.GET)
16     public ModelAndView handleRequest(HttpServletRequest request,
17         HttpServletResponse response)
18         throws ServletException, IOException {
19         ModelAndView modelview = new ModelAndView("welcome");
20         modelview.getModelMap().addAttribute("benvinguda", "Benvingut Estoc de
21         Medicaments!");
22         modelview.getModelMap().addAttribute("tagline", "Una aplicació de l'
23         Institut Obert de Catalunya");
24         return modelview;
25     }
26 }

```

Finalment, creeu la carpeta *views* dins de *WEB-INF* i a dins el fitxer `welcome.jsp` amb el contingut que es mostra a continuació.

```

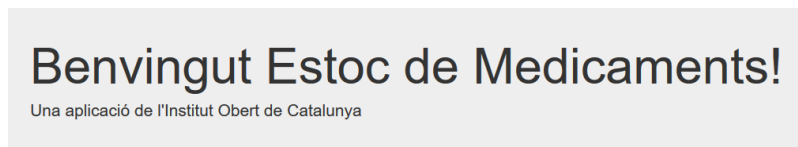
1 <%@ page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>

```

```
3 <html lang="ca">
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/
7       bootstrap.min.css">
8     <title>Welcome</title>
9   </head>
10  <body>
11    <section>
12      <div class="jumbotron">
13        <div class="container">
14          <h1> ${benvinguda} </h1>
15          <p> ${tagline} </p>
16        </div>
17      </div>
18    </section>
19  </body>
</html>
```

En executar l'aplicació us ha d'aparèixer el bàner que es mostra en la figura 1.14.

FIGURA 1.14. Sortida de l'aplicació. "Estoc de medicaments", benvinguda



Ara ja teniu el projecte amb els elements i la configuració bàsica per continuar desenvolupant el vostre estoc de medicaments.

1.5 Què s'ha après?

En aquest apartat hem fet servir **Maven** com a eina que dóna estructura als projectes i permet baixar les llibreries que cal fer servir des de repositoris centrals a partir de la configuració de **dependències**. El gran avantatge de Maven és fer independent el nostre desenvolupament de l'IDE que es fa servir.

Hem introduït el procés **petició-resposta** de Spring MVC, així com a l'arquitectura que en dóna suport. Heu creat dues aplicacions Spring MVC amb la mateixa estructura, és a dir, una part de vistes .jsp, la configuració del Dispatcher Servlet (o Front Controller) i la classe que fa de controlador per manejar les peticions i que retorna l'objecte ModelAndView.

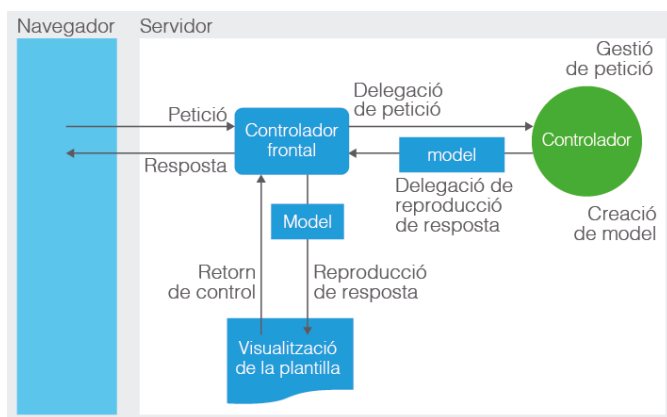
Les aplicacions fetes fins ara en aquesta unitat només mostren un text variable o no, encara que amb l'arquitectura de Spring MVC. Hem d'avançar una mica més i donar més funcionalitat a "Estoc de medicaments".

En els apartats següents veurem com afegir aquesta funcionalitat de manera ordenada, creant els paquets i les classes amb els patrons que suggereix Spring MVC.

2. Spring MVC, aplicació web

L'arquitectura i el procés d'Spring MVC respecte a una petició des d'un navegador és la que es mostra a la figura 2.1.

FIGURA 2.1. Arquitectura i procés d'una aplicació Spring MVC



La petició feta des d'un navegador client és recollida pel Front Controller (Dispatcher Servlet) i la passa al controlador adient. Aquest construeix el **model** i el retorna (amb l'estat que correspongui) al Dispatcher Servlet que retornarà la resposta, determinant la vista a retornar a partir del View Resolver assignat i amb els valors del model.

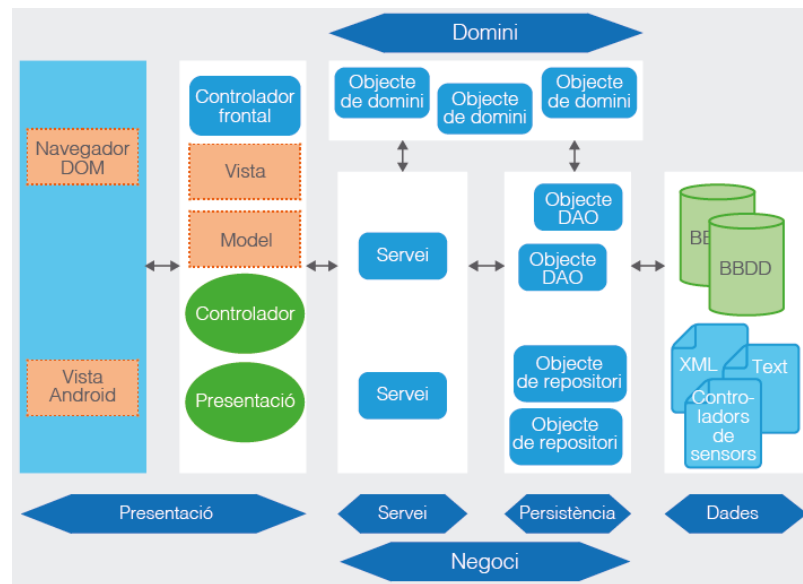
Veurem com el controlador construeix el model, és a dir, la part de negoci de la vostra aplicació web (*enterprise-level*).

Amb Spring MVC podeu fer servir una bona pràctica per a aquest tipus de desenvolupament, consistent a estructurar el codi en capes (*layers*) i donant reusabilitat i baix acoblament a la vostra aplicació.

Les capes que es recomanen són quatre:

- presentació (*Presentation*)
- domini (*Domain*)
- serveis (*Services*)
- persistència (*Persist*)

El diagrama de la figura 2.2 mostra aquestes capes i les seves interaccions.

FIGURA 2.2. Capes i interaccions d'una aplicació Spring MVC

Els objectes vistos fins ara, com `Dispatcher Servlet`, controladors, `View Resolvers` i d'altres conformen la capa de **presentació**.

La capa de **persistència** és la que conté els objectes que interaccionen amb les dades; per exemple, obtenint un conjunt de registres d'una base de dades a partir d'una consulta SQL.

Un controlador podria demanar directament les dades a la capa de persistència, però a l'arquitectura del diagrama es proposa la creació de la capa de **servei** per tal de poder aplicar les regles de negoci amb i/o abans d'obtenir les dades. Per exemple, l'aplicació d'una restricció no implementada a la base de dades, com no deixar a un client fer una comanda si el seu deute és superior a cert import.

En tot cas, les dades que s'obtenen o s'estan construint es mapegen sobre els objectes de la capa de **domini**; per exemple, objectes de classes entitat que poden representar una taula d'una base de dades.

Continuareu amb el vostre projecte d'Estoc de medicaments ("stmedioc") desenvolupant la resta de capes i descrivint els conceptes i la metodologia implicades en cadascuna d'elles.

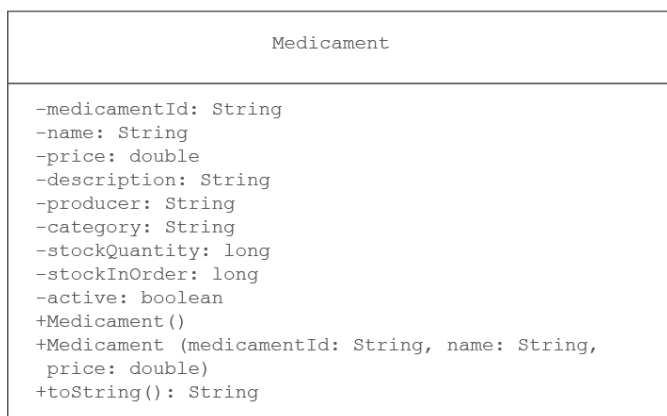
2.1 Estoc de medicaments, capa de domini

La capa **domini** d'una aplicació web consisteix en la implementació de les classes d'un o més models de domini.

El **model de domini** és la representació, per exemple amb un diagrama UML, de les classes corresponents a les dades del problema a resoldre de la lògica de negoci.

En el cas de l'aplicació d'Estoc de medicaments, dissenyada amb propòsits pedagògics, només hi ha una entitat *Medicament* i, per tant, la capa de domini només tindrà una classe, que anomenarem *Medicament* (vegeu la figura 2.3).

FIGURA 2.3. Classe *Medicament*



Un entitat *Medicament* podria contenir més propietats i mètodes, però de moment només fareu servir aquest a la vostra aplicació. El nom de les propietats és suficient per relacionar-les amb els conceptes que representen. No obstant això, cal fer algunes apreciacions:

- *medicamentID* és el codi de medicament.
- *producer* és el proveïdor.
- *stockQuantity* són les unitats emmagatzemades i *stockInOrder* són les unitats que estan demanades al proveïdor però encara no heu introduït al magatzem.
- *active* és per indicar si un medicament es fa servir (valor *true*) o s'ha donat de baixa (valor *false*).

Els objectes creats amb classes de domini s'acostumen a dir **objectes de domini** (*Domain Object*).

2.1.1 Creació del domini

Com que el nostre domini està format únicament per la classe *Medicament*, al projecte "stmedioc" heu de crear un paquet `cat.xtec.ioc.domain` i la classe *Medicament* amb el contingut que es mostra a continuació:

```

1 package cat.xtec.ioc.domain;
2
3 public class Medicament {
4
5     private String medicamentId;
6     private String name;

```

El codi del projecte "stmedioc" en l'estat de capa de domini es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament de la capa de domini és millor partir del que ja heu fet servir per a Estoc de medicaments. Benvinguda i copiar-lo amb el nom de "stmedioc201".

```
7     private double price;
8     private String description;
9     private String producer;
10    private String category;
11    private long stockQuantity;
12    private long stockInOrder;
13    private boolean active;
14
15    public Medicament() {
16        super();
17    }
18
19    public Medicament(String medicamentId, String name, double price) {
20        this.medicamentId = medicamentId;
21        this.name = name;
22        this.price = price;
23    }
24
25    //Heu d'afegir els getters i setters de totes les propietats
26
27    @Override
28    public boolean equals(Object obj) {
29        if (this == obj) {
30            return true;
31        }
32        if (obj == null) {
33            return false;
34        }
35        if (getClass() != obj.getClass()) {
36            return false;
37        }
38        Medicament other = (Medicament) obj;
39        if (medicamentId == null) {
40            if (other.medicamentId != null) {
41                return false;
42            }
43        } else if (!medicamentId.equals(other.medicamentId)) {
44            return false;
45        }
46        return true;
47    }
48
49    @Override
50    public int hashCode() {
51        final int prime = 31;
52        int result = 1;
53        result = prime * result
54            + ((medicamentId == null) ? 0 : medicamentId.hashCode());
55        return result;
56    }
57
58    @Override
59    public String toString() {
60        return "Medicament [codi=" + medicamentId + ", nom=" + name + "];"
61    }
62 }
```

2.1.2 Fent servir el domini

Penseu ara en la necessitat de mostrar per al navegador un medicament mitjançant una petició GET des d'un URL.

Davant d'una petició, el Dispatcher Servlet delegarà a un controlador la petició i aquest s'encarregarà de construir el model.

A Spring MVC, amb l'estructura de capes que esteu desenvolupant, implementareu un nou controlador `MedicamentController` que agafi aquesta petició i construeixi el model a partir del domini (de la classe `Medicament`).

Heu de crear la classe `MedicamentController` dins del paquet `cat.xtec.ioc.controller` amb el contingut que es mostra a continuació.

```
1 package cat.xtec.ioc.controller;
2
3 import cat.xtec.ioc.domain.Medicament;
4 import java.io.IOException;
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RequestMethod;
11 import org.springframework.web.servlet.ModelAndView;
12
13 @Controller
14 public class MedicamentController {
15
16     @RequestMapping(value = "/medicaments", method = RequestMethod.GET)
17     public ModelAndView handleRequest(HttpServletRequest request,
18                                     HttpServletResponse response)
19         throws ServletException, IOException {
20         ModelAndView modelview = new ModelAndView("medicaments");
21         Medicament ibuprofe = new Medicament("M010", "Ibuprofé", 2);
22         ibuprofe.setDescription("Ibuprofé de 600mg");
23         ibuprofe.setCategory("Anti-inflamatori");
24         ibuprofe.setProducer("Cinfa");
25         ibuprofe.setStockQuantity(214);
26         modelview.getModelMap().addAttribute("medicament", ibuprofe);
27         return modelview;
28     }
29 }
```

Cal destacar que al controlador `MedicamentController` esteu donant valors a l'entitat, i en el model de capes això serà responsabilitat d'altres. En aquest cas, doneu els valors en el controlador perquè encara no heu desenvolupat les altres capes, però després canviareu aquesta classe per fer-ho on toca.

Fixeu-vos que fins ara els atributs que heu afegit al model eren simples, com *string*. En el cas de `MedicamentController` esteu creant un atribut que és un objecte: un *domain object* `Medicament`.

`MedicamentController` retorna un `ModelAndView` amb nom *medicaments*. Segons la configuració que havíeu fet per al View Resolver a `DispatcherServlet-servlet.xml`, Spring cercarà la vista `WEB-INF/views/medicaments.jsp` per mostrar-la i afegirà l'atribut `medicament` (objecte `Medicament`) a la resposta.

Heu de crear aquesta vista a la carpeta esmentada amb el contingut que es mostra a continuació.

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html lang="ca">
4     <head>
5         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6         <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/
7             css/bootstrap.min.css">
8         <title>Medicaments</title>
```

```

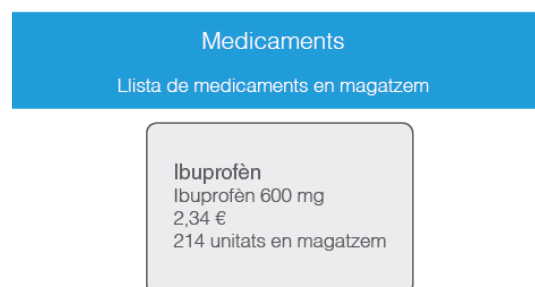
8      </head>
9      <body>
10         <section>
11             <div class="jumbotron">
12                 <div class="container">
13                     <h1>Medicaments</h1>
14                     <p>Llista de medicaments en magatzem</p>
15                 </div>
16             </div>
17         </section>
18         <section class="container">
19             <div class="row">
20                 <div class="col-sm-6 col-md-3" style="padding-bottom: 15px">
21                     <div class="thumbnail">
22                         <div class="caption">
23                             <h3>${medicament.name}</h3>
24                             <p>${medicament.description}</p>
25                             <p>${medicament.price} €</p>
26                             <p>Hi ha ${medicament.stockQuantity} unitats en
27                                 magatzem</p>
28                         </div>
29                     </div>
30                 </div>
31             </section>
32         </body>
33     </html>

```

Fixeu-vos que esteu servant les expressions tipus EL (*Expression Language*) `${atribut.propietat}` i que com a atribut heu passat un objecte `Medicament`, i per tant podeu usar les seves propietats. Tingueu en compte que, per exemple, amb `${medicament.name}` s'executarà el mètode `medicament.getName()`, i si no l'heu implementat (*getters and setters*) llavors el resultat serà `null`.

Si executeu l'aplicació i afegiu a l'URL del navegador `/medicaments` podreu veure el que es mostra en la figura 2.4.

FIGURA 2.4. Sortida de l'aplicació Estoc de medicaments, domini



Partíeu de la capa de presentació feta, però anireu afegint controladors i vistes a mesura que les necessiteu.

La capa de **domini** es correspon pràcticament amb les classes Entitat del problema a resoldre, i en el cas de l'aplicació Estoc de medicaments és la classe `Medicament`. Però heu fet trampes posant valors en el controlador `MedicamentController` amb la finalitat de provar. I això s'ha de resoldre continuant amb el desenvolupament de les següents capes.

2.2 Estoc de medicaments, capa de persistència

La capa de **persistència** és el conjunt d'objectes que interaccionen amb les dades. Les dades poden estar emmagatzemades en bases de dades i en altres tipus de formats, com XML, JSON, imatges, vídeo, etc.

La capa de persistència conté objectes **repositori** (*Repository Objects*) que permeten mapejar les dades de la font de dades (base de dades o no) amb els objectes de domini (*Domain Object*). Són, en realitat, els responsables de les operacions *CRUD* (*Create, Read, Update i Delete*).

En operacions d'obtenció de dades, els objectes repositori consulten la font de dades, per exemple via SQL, i el resultat és mapejat sobre objectes de domini.

En operacions d'actualització de les dades, els objectes repositori parteixen de l'estat dels objectes de domini i amb els seus valors actualitzen la font de dades.

Per indicar a Spring que una classe és un repositori es fa servir l'anotació `@Repository` (`org.springframework.stereotype.Repository`). Aquesta anotació també permet que les excepcions SQL es converteixin en `DataAccessExceptions` de Spring.

Amb aquests conceptes descrits ja podeu crear la capa de persistència de la vostra aplicació Estoc de medicaments.

Al desenvolupament de la capa de domini heu mostrat un únic medicament que, a més, temporalment, es crea directament al controlador. Mostrareu una llista de medicaments i a més traureu la creació del medicament del controlador.

La manera d'accedir a les dades està fora de l'abast d'aquesta unitat, i per això fareu servir objectes en memòria com si fossin la base de dades. Només haureu de canviar aquests objectes per d'altres que realment es connectin i dialoguin amb la base de dades si voleu dotar de veritable persistència l'Estoc de medicaments.

Creareu una interfície `MedicamentRepository` i la implementareu amb la classe `InMemoryMedicamentRepository` amb els mètodes per mostrar la llista de medicaments, només un mètode en aquest cas. Per altra banda, fareu que el controlador dialogui directament amb aquesta capa de persistència, encara que ho canviareu més endavant per tal que dialogui amb la capa de servei.

Feu servir una interfície, perquè això us permet que el controlador declari i cridi la interfície, i d'aquesta manera podeu canviar la implementació sense canviar res més.

2.2.1 Creació de la persistència

Afegiu el paquet *cat.xtec.ioc.domain.repository* al nostre projecte “stmedioc”, i en aquest paquet una nova interfície *MedicamentRepository* amb el contingut següent:

```

1 package cat.xtec.ioc.domain.repository;
2
3 import cat.xtec.ioc.domain.Medicament;
4 import java.util.List;
5
6 public interface MedicamentRepository {
7     List <Medicament> getAllMedicaments();
8 }

```

El codi del projecte “stmedioc” en l'estat de capa de persistència es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament de la capa de persistència és millor partir del que ja heu fet servir per a Estoc de medicaments, capa de domini, i copiar-lo amb el nom de “stmedioc202”.

Afegiu el paquet *cat.ioc.xtec.domain.repository.impl* al nostre projecte “stmedioc”. També us interessa que Spring cerqui automàticament les classes d'aquest paquet per poder injectar els objectes quan es trobi una anotació d'aquest tipus. Per això, canvieu la propietat `context:component-scan` de *DispatcherServlet-servlet.xml*:

```

1 <context:component-scan base-package="cat.xtec.ioc.controller cat.xtec.ioc.
   domain.repository" />

```

En el paquet *cat.ioc.xtec.domain.repository.impl*, la classe *InMemoryMedicamentRepository* implementa *MedicamentRepository* amb el següent contingut:

```

1 package cat.xtec.ioc.domain.repository.impl;
2
3 import cat.xtec.ioc.domain.Medicament;
4 import cat.xtec.ioc.domain.repository.MedicamentRepository;
5 import java.util.ArrayList;
6 import java.util.List;
7 import org.springframework.stereotype.Repository;
8
9 @Repository
10 public class InMemoryMedicamentRepository implements MedicamentRepository {
11
12     private List<Medicament> listOfMedicaments = new ArrayList<Medicament>();
13
14     public InMemoryMedicamentRepository() {
15         Medicament ibuprofe = new Medicament("M010", "Ibuprofé", 2);
16         ibuprofe.setDescription("Ibuprofé de 600mg");
17         ibuprofe.setCategory("Anti-inflamatori");
18         ibuprofe.setProducer("Cinfa");
19         ibuprofe.setStockQuantity(214);
20
21         Medicament paracetamol = new Medicament("M020", "Paracetamol", 2.6);
22         paracetamol.setDescription("Paracetamol 1g");
23         paracetamol.setCategory("Analgèsic");
24         paracetamol.setProducer("Ferrer");
25         paracetamol.setStockQuantity(56);
26
27         Medicament acacetilsalicilico = new Medicament("M030", "Ac Acetil Salicilico", 2.6);
28         acacetilsalicilico.setDescription("Ac Acetil Salicílico");
29         acacetilsalicilico.setCategory("Analgèsic");
30         acacetilsalicilico.setProducer("Bayer");
31         acacetilsalicilico.setStockQuantity(15);

```

```
32     listOfMedicaments.add(ibuprofe);
33     listOfMedicaments.add(paracetamol);
34     listOfMedicaments.add(acacetilsalicilico);
35 }
36
37
38 public List<Medicament> getAllMedicaments() {
39     return listOfMedicaments;
40 }
41 }
```

Aquesta classe repository hauria de connectar amb la font de dades, però com que és fora de l'abast d'aquesta unitat feu que les dades resideixin en memòria creant-les en el constructor. Per això, `getAllMedicaments` retorna la llista de medicaments propietat de la classe, però en realitat hauria d'anar a buscar-la a la font de dades.

Canvieu el controlador `MedicamentController` per obtenir la llista de medicaments des de la capa de persistència. El contingut de `MedicamentController` sense els imports és com es mostra a continuació:

```
1 @Controller
2 public class MedicamentController {
3
4     @Autowired
5     private MedicamentRepository medicamentRepository;
6
7     @RequestMapping(value = "/medicaments", method = RequestMethod.GET)
8     public ModelAndView handleRequest(HttpServletRequest request,
9                                     HttpServletResponse response)
10         throws ServletException, IOException {
11         ModelAndView modelview = new ModelAndView("medicaments");
12         modelview.getModelMap().setAttribute("medicaments",
13                                             medicamentRepository.getAllMedicaments());
14         return modelview;
15     }
16 }
```

Haureu d'afegir-hi els imports:

```
1 import org.springframework.beans.factory.annotation.Autowired;
2 import cat.xtec.ioc.domain.repository.MedicamentRepository;
```

Hi podeu suprimir l'import:

```
1 import cat.xtec.ioc.domain.Medicament;
```

S'ha creat la propietat `medicamentRepository` que, mitjançant l'anotació `@Autowired`, serà inicialitzada per Spring amb l'objecte de tipus `MedicamentRepository` del Web Application Context.

S'ha canviat la creació d'un únic medicament en la mateixa classe controlador per a la crida a la llista de tots els medicaments que us proporciona la persistència. Noteu que s'ha canviat l'atribut `medicament` per `medicaments`.

Fixeu-vos com no es crida directament la classe `InMemoryMedicamentRepository`, ja que es declara i es crida la interfície.

Això us permetrà canvis en la font de dades i en la manera de dialogar amb ella sense tocar el controlador.

Només us queda mostrar la llista dels medicaments que hi ha a la vostra font de dades (creats en memòria). Canvieu la vista `medicaments.jsp`: heu d'afegir al principi de tot la declaració del `taglib` i heu de canviar tot el contingut de la `div` amb classe="row" pel codi que es mostra a continuació.

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 ...
3         <c:forEach items="${medicaments}" var="medicament">
4             <div class="col-sm-6 col-md-3" style="padding-bottom: 15px">
5                 <div class="thumbnail">
6                     <div class="caption">
7                         <h3>${medicament.name}</h3>
8                         <p>${medicament.description}</p>
9                         <p>${medicament.price}</p>
10                        <p>Hi ha ${medicament.stockQuantity} unitats en
11                          magatzem</p>
12                    </div>
13                </div>
14            </c:forEach>

```

Executeu l'aplicació i afegiu `/medicaments` a l'URL. Heu d'obtenir un resultat similar al de la figura 2.5.

FIGURA 2.5. Sortida de l'aplicació Estoc de medicaments, persistència

Medicaments		
Llista de medicaments en magatzem		
Ibuprofèn Ibuprofèn 600 mg 2,0 € 214 unitats en magatzem	Paracetamol Paracetamol 1 g 2,6 € 56 unitats en magatzem	Àcid acetilsalílic Àcid acetilsalílic 2,6 € 15 unitats en magatzem

L'URL `/medicaments` ha fet que el Dispatcher Servlet passi la petició a `MedicamentController`. En primer lloc, demana la injecció d'un objecte `MedicamentRepository` (anotació `@Autowired`) i Spring retorna un objecte de la implementació `InMemoryMedicamentRepository`.

Recordeu que feu servir una classe que construeix valors ficticis en memòria perquè l'accés a dades no és l'objectiu d'aquesta unitat, però podríeu fer una classe que retornarà els valors des d'una base de dades, per exemple `MedicamentDAO`, que implementaria `MedicamentRepository` i no hauríeu de canviar `MedicamentController`. Això és l'acoblament feble entre capes.

En segon lloc, es construeix el `ModelAndView` a partir del repositori, demanant tots els medicaments al repositori i passant-los a la vista `Medicaments` sobre l'atribut del mateix nom.

Finalment, es mostra la vista que, amb etiquetes de JSTL, recorrerà els objectes de l'atribut `medicaments` i els mostrarà.

2.3 Estoc de medicaments, capa de servei

El projecte Estoc de medicaments conté la capa de presentació, amb les vistes i els controladors; la capa de domini, amb la representació d'un medicament, i la capa de persistència, que dialoga amb les dades mitjançant objectes `repository`.

En el projecte actual, els controladors criden directament, via injecció, els objectes `repository`. No obstant això, si recupereu el diagrama d'arquitectura i procés d'una aplicació Spring MVC, els controladors només dialoguen amb la capa de servei que us falta desenvolupar.

Els objectes `repository` fan operacions *CRUD* simples, i els resultats els poden emmagatzemar en memòria en forma d'objectes de domini. Heu vist que obteniu una llista de medicaments perquè està guardada en una *List* d'objectes `Medicament` (objectes de domini) al repositori.

Però on posareu operacions que siguin més complexes que abastin més d'una operació *CRUD*, més d'una entitat, o simplement que afegixin restriccions? És a dir, on posareu les regles de negoci?

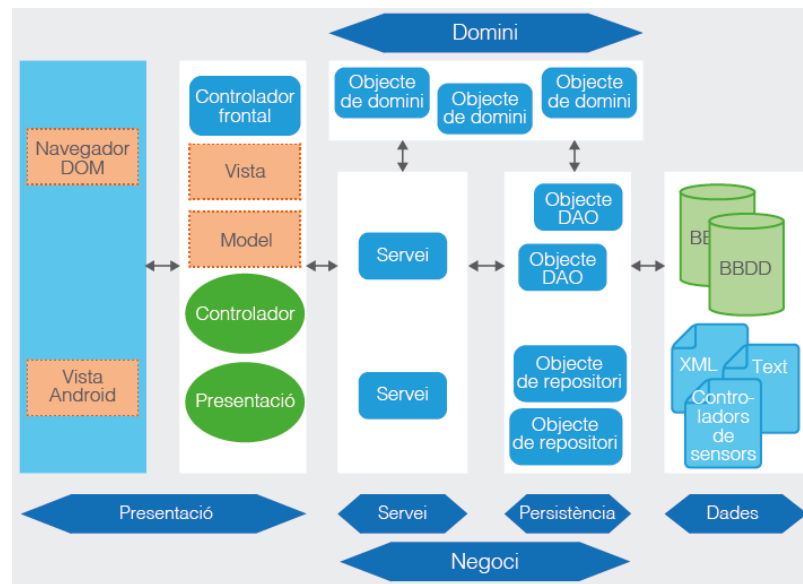
Per exemple, penseu en un moviment d'estoc del nostre medicament que signifiqui baixar 10 unitats d'ibuprofèn. En una primera aproximació podríeu dir que això consisteix a crear un mètode al repositori que faci un *update* (com és a memòria, seria actualitzar l'element adient a `List<Medicament> listOfMedicaments` de `InMemoryMedicamentRepository`).

Però hauríeu de fer més operacions. Hauríeu de veure que el medicament que es vol actualitzar existeix, que té suficient estoc, i després fer efectiva l'actualització. No ho fareu per ara, però a tot això hauríeu d'afegir-li la comprovació de si l'usuari té autorització per fer aquesta operació.

Per contenir aquestes operacions complexes que abasten més d'una operació simple, on podeu fer servir un o més objectes de domini, fareu servir la capa de **servei**.

Com podeu veure en apartats d'accés a dades, les transaccions es defineixen a la **capa de servei**.

En el projecte **Estoc de medicaments** que esteu desenvolupant creareu un servei que faci els moviments d'estoc sobre els medicaments, és a dir, que simplement incrementi o redueixi les unitats en estoc. Llavors, la crida de la capa de presentació es farà com preveu l'arquitectura i el procés d'una aplicació web amb Spring MVC, és a dir, com es mostra en la figura 2.6.

FIGURA 2.6. Capes i interaccions a Spring MVC

Amb aquest objectiu, creareu un controlador específic que atengui les peticions de moviments d'estoc que cridi el vostre servei i que retorni el resultat. El servei cridarà el repositori i aquest farà l'actualització. Com és a memòria, farà l'actualització sobre la *List* de medicaments.

Però al repositori només hi ha un mètode que retorna la llista de tots els medicaments. Llavors heu de modificar el repositori per adaptar-lo a les operacions que havíeu enumerat:

- El medicament que es vol actualitzar existeix.
- El medicament té suficient estoc (només si és un moviment de decrement).
- Fer efectiva l'actualització.

Heu d'afegir al repositori els mètodes adients per resoldre les operacions.

El codi del projecte "stmedioc" en l'estat de capa de servei es pot descarregar del següent .

Però per seguir el desenvolupament de la capa de servei és millor partir del que ja heu fet servir per a Estoc de medicaments, capa de persistència, i copiar-lo amb el nom de "stmedioc203" del següent .

2.3.1 Adaptació del repositori

Heu d'accedir a un medicament concret a partir del seu codi (`medicamentId`), i això us permetrà comprovar que existeix i veure si es pot fer el moviment d'estoc (ha de tenir prou unitats si voleu treure estoc).

Modifiqueu la interfície `MedicamentRepository` afegint-hi el mètode `Medicament getMedicamentById(String medicamentId);`.

Implementeu el mètode a `InMemoryMedicamentRepository` tal com es mostra a continuació:

```
1 public Medicament getMedicamentById(String medicamentId) {
2     Medicament medicamentById = null;
3     for (Medicament medicament : listOfMedicaments) {
4         if (medicament != null && medicament.getMedicamentId() != null
5             && medicament.getMedicamentId().equals(medicamentId)) {
6             medicamentById = medicament;
7             break;
8         }
9     }
10    if (medicamentById == null) {
11        throw new IllegalArgumentException(
12            "No s'han trobat medicaments amb el codi: " + medicamentId)
13        ;
14    }
15    return medicamentById;
16 }
```

El mètode `getMedicamentById` cerca el medicament que coincideix en *id* amb el codi que es passa pel paràmetre. Fixeu-vos que si el troba retorna l'objecte de domini `Medicament`, i en un altre cas, llança una excepció.

2.3.2 Creació del servei

Heu de crear el servei que faci totes les operacions per aconseguir l'actualització de l'estoc comprovant les regles de negoci: existeix el medicament i no deixeu l'estoc en negatiu.

Com en altres casos fareu servir una interfície, ja que us permet l'acoblament més feble entre les capes.

Creeu un nou paquet `cat.xtec.ioc.service` i en ell la interfície `MovimentStockService` amb el codi que es mostra a continuació:

```
1 package cat.xtec.ioc.service;
2
3 public interface MovimentStockService {
4     void processMovimentStock(String medicamentId, long quantity, int signe);
5 }
```

Creeu un nou paquet `cat.xtec.ioc.service.impl` i també la implementació `MovimentStockServiceImpl` amb el codi que es mostra a continuació:

```
1 package cat.xtec.ioc.service.impl;
2
3 import cat.xtec.ioc.domain.Medicament;
4 import cat.xtec.ioc.domain.repository.MedicamentRepository;
5 import cat.xtec.ioc.service.MovimentStockService;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
```

```
9 @Service
10 public class MovimentStockServiceImpl implements MovimentStockService {
11
12     @Autowired
13     private MedicamentRepository medicamentRepository;
14
15     public void processMovimentStock(String medicamentId, long quantity, int
        signe) {
16         Medicament medicamentById = medicamentRepository.getMedicamentById(
            medicamentId);
17         long signedQuantity = quantity * signe;
18         if ((medicamentById.getStockQuantity() + signedQuantity) < 0) {
19             throw new IllegalArgumentException("No hi ha prou unitats. La
                quantitat en estoc és: " + medicamentById.getStockQuantity());
20         }
21         medicamentById.setStockQuantity(medicamentById.getStockQuantity() +
            signedQuantity);
22     }
23 }
```

L' anotació `@Autowired` fa que Spring us retorni un objecte sense necessitat de crear-lo directament al nostre codi. En aquest cas volem fer servir `MedicamentRepository`.

El mètode `processMovimentStock` fa el conjunt d'operacions que requeríem. Demana al repositori el medicament i fa l'actualització del seu estoc amb `setStockQuantity`. Però en cap cas deixarà l'estoc en negatiu, perquè abans llançaria l'excepció `IllegalArgumentException`. Recordeu que al mètode `getMedicamentById` també es llança una excepció en cas de no existir el medicament.

També us interessa que Spring cerqui automàticament les classes d'aquest paquet per poder injectar els objectes quan es trobi una anotació d'aquest tipus. Per això, canvieu la propietat `context:component-scan` de `DispatcherServlet-servlet.xml`:

```
1 <context:component-scan base-package="cat.xtec.ioc.controller cat.xtec.ioc.
    domain.repository cat.xtec.ioc.service" />
```

2.3.3 Afegits a la capa de presentació

En comptes de fer servir `MedicamentController` emprareu un nou controlador específic per a aquest cas.

Al paquet `cat.xtec.ioc.controller`, creeu el controlador `MovimentStockController` amb el codi següent:

```
1 package cat.xtec.ioc.controller;
2
3 import cat.xtec.ioc.service.MovimentStockService;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Controller;
6 import org.springframework.web.bind.annotation.RequestMapping;
7
8 @Controller
9 public class MovimentStockController {
```

```
10
11 @Autowired
12 private MovimentStockService movimentStockService;
13
14 @RequestMapping("/movimentestoc/M020/2/-1")
15 public String process() {
16     movimentStockService.processMovimentStock("M020", 2, -1);
17     return "redirect:/medicaments";
18 }
19 }
```

Amb l'URL de `@RequestMapping`, el controlador crida la capa de servei per fer el moviment d'estoc. Si hi ha cap excepció es mostrarà el missatge que toca, però si no mostrarà la llista de medicaments on podeu veure el decrement de l'estoc (signe -1 en l'exemple).

En aquest cas, l'URL és fix i es crida a fer el moviment amb constants. No us preocupeu, ja veureu com fer tot això variable.

Fins ara heu fet servir la creació explícita del `ModelAndView` com a resposta, però també es pot retornar un *string* amb el nom de la vista i Spring crearà l'objecte `ModelAndView` per a vosaltres.

En el cas que us ocupa heu d'anar a la vista `Medicaments`, i per evitar tornar a fer el moviment d'estoc, si l'usuari prem el botó *Anar enrere* del navegador, feu una redirecció (return `"redirect:/medicaments"`). I per fer servir `redirect` heu emprat el tipus *string* com a retorn.

Executeu l'aplicació i aneu a `/medicaments`. El navegador us mostra la llista de medicaments amb l'estoc actual (vegeu la figura 2.7).

FIGURA 2.7. Sortida de l'aplicació Estoc de medicaments, estoc actual

Medicaments		
Lista de medicaments en magatzem		
Ibuprofèn Ibuprofèn 600 mg 2,0 € 214 unitats en magatzem	Paracetamol Paracetamol 1 g 2,6 € 56 unitats en magatzem	Àcid acetilsalicílic Àcid acetilsalicílic 2,6 € 15 unitats en magatzem

Ara canvieu l'URL a `/movimentestoc/M020/2/-1` i veureu el següent resultat, que mostra com ha variat l'estoc del paracetamol (vegeu la figura 2.8).

FIGURA 2.8. Sortida de l'aplicació Estoc de medicaments, estoc modificat

Medicaments		
Lista de medicaments en magatzem		
Ibuprofèn Ibuprofèn 600 mg 2,0 € 214 unitats en magatzem	Paracetamol Paracetamol 1 g 2,6 € 54 unitats en magatzem	Àcid acetilsalicílic Àcid acetilsalicílic 2,6 € 15 unitats en magatzem

2.3.4 Què heu fet i què heu après amb la capa de servei?

A l'aplicació Estoc de medicaments heu implementat un moviment d'estoc d'un medicament, comprovant que existeix i que no deixa l'estoc en negatiu.

Per fer això heu seguit les recomanacions d'Spring MVC i heu creat la capa de **servei**, on s'implementen regles de negoci, és a dir, operacions complexes que poden abastar una o més entitats i que poden contenir operacions simples.

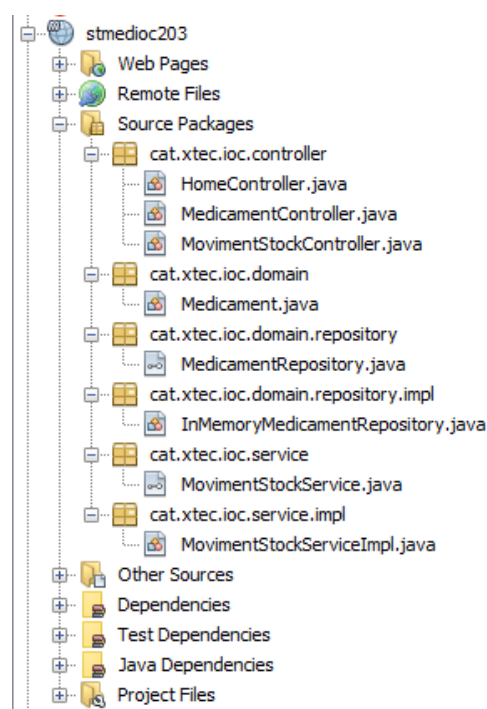
Heu creat un nou controlador que crida la capa de servei, i és aquesta capa la que crida la de persistència. A la vegada, heu afegit els mètodes necessaris a la persistència per complir amb les noves peticions. En el vostre cas, la persistència us proporciona un medicament mitjançant el codi.

2.4 Què s'ha après?

A Estoc de medicaments partíeu d'una pàgina de benvinguda, i ara podeu mostrar la llista de medicaments i podeu fer moviments d'estoc (encara de manera fixa).

Per fer això heu estructurat el projecte en capes, tal com es mostra en la figura 2.9.

FIGURA 2.9. Estructura de capes d'una aplicació Spring MVC



Heu seguit l'estructuració de codi recomanada per a aplicacions web amb Spring MVC amb les capes següents:

- presentació (*Presentation*)

- domini (*Domain*)
- servei (*Service*)
- persistència (*Persistence*)

Heu vist que la capa de **presentació** la conformen les vistes jsp i les classes controladores. A més, mitjançant fitxers de configuració i anotacions, Spring MVC afegeix altres classes necessàries, com el Dispatcher Servlet i View Resolvers.

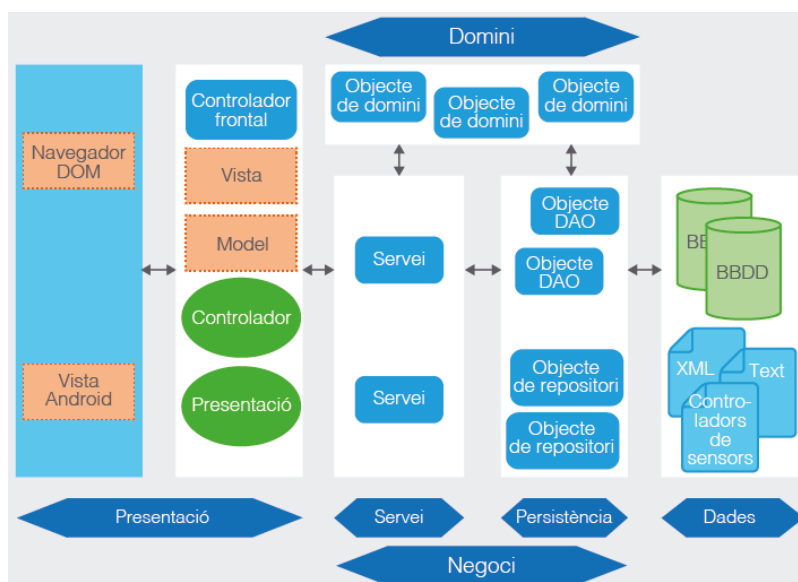
Heu desenvolupat la **persistència** amb objectes repositori que dialoguen amb les dades. En el vostre cas no accediu a una base de dades, perquè no és dins de l'abast d'aquesta unitat; treballeu amb dades creades en memòria.

Heu creat la capa **domini** com la representació del nostre model de dades, una classe per a cada entitat. En el vostre cas, una única classe *Medicament*.

Heu acabat amb la capa de **servei**, que implementa les regles de negoci. En el vostre cas, un moviment d'estoc que es compon de diverses operacions: comprovació de l'existència del medicament, verificar que l'estoc no queda en negatiu i finalment, actualitzar les unitats en estoc.

Heu connectat tot segons l'arquitectura que proposa Spring MVC (vegeu la figura 2.10).

FIGURA 2.10. Capes i interaccions a Spring MVC



Heu fet injecció de dependències i l'heu fet mitjançant interfícies per aconseguir un acoblament feble entre les capes que permet un millor manteniment i adaptació futurs.

L'aplicació Estoc de medicaments us mostra una llista de medicaments, i podeu fer un moviment d'estoc. Però us animo a continuar amb aquests materials per descobrir com fer que el moviment d'estoc sigui variable i moltes coses més, com crear productes amb formularis des del navegador.

3. Spring MVC, aprofundint en els controladors

Partiu de l'aplicació web **Estoc de medicaments** que mostra una pàgina de benvinguda, i que mitjançant l'URL adient pot mostrar una llista de medicaments i fins i tot rebaixar l'estoc d'un d'aquests.

Aquesta aplicació segueix l'arquitectura i el procés de qualsevol aplicació Spring MVC. L'aplicació s'estructura en les quatre capes següents:

- presentació (*Presentation*)
- domini (*Domain*)
- servei (*Service*)
- persistència (*Persistence*)

Estoc de medicaments fa servir anotacions de Spring MVC:

- `@Controller`, `@Repository` i `@Service`, que defineixen el tipus d'objecte Spring.
- `@RequestMapping`, que caça l'URL definit i executa el mètode que el segueix.
- `@Autowired`, que injecta l'objecte que segueix.

A més, Estoc de medicaments conté els fitxers XML amb la configuració necessària per dirigir el comportament de Spring MVC respecte a l'aplicació.

Anem a afegir més funcionalitat a l'aplicació Estoc de medicaments per conèixer els conceptes i procediments associats, aprofundint un punt més en l'estudi dels controladors.

Estoc de medicaments mostra la llista dels medicaments. El procés per mostrar la llista el canviareu per adaptar-lo a un veritable procés Spring MVC, és a dir, creant el servei corresponent i fent que el controlador cridi el servei en comptes del que fa ara, cridar directament la persistència. Aprofitant aquesta implementació, usareu l'anotació `@RequestMapping` a nivell de classe i a nivell de mètode.

Fareu que Estoc de medicaments mostri medicaments d'una categoria, però en aquest cas introduïreu com l'URL de `@RequestMapping` pot ser variable (*path variable*) afegint l'anotació `@PathVariable`. Ja us convé, atès que ara, pel moviment d'estoc, l'URL és fix.

Aprofundint en variables a l'URL permetreu que Estoc de medicaments mostri medicaments basats en un **filtre**, és a dir, en un conjunt de parells (propietat,

valor) que implementareu amb l'anotació `@MatrixVariable` i una col·lecció *Map* (*matrix variable*).

Finalment, l'aplicació podrà mostrar el detall d'un medicament. En aquest cas no fareu servir variables a l'URL, sinó que utilitzareu l'anotació `@RequestParam` per definir paràmetres HTTP a partir de peticions Get o Post.

3.1 Estoc de medicaments, capa de servei a medicament

En l'aplicació Estoc de medicaments de la qual partiu, el controlador `MedicamentController` crida directament la capa de persistència; concretament, es fa un injecció de `MedicamentRepository`.

Arregleu això implementant la capa de servei corresponent i aprofiteu per endreçar una mica les crides fent que mostrar tots els medicaments sigui un cas particular de mostrar medicaments.

El codi del projecte "stmedioc" en l'estat capa de servei a medicament es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament que segueix és millor partir del que ja heu fet servir i copiar-lo amb el nom de "stmedioc301".

3.1.1 Creant el servei a medicament

Al paquet `cat.xtec.ioc.service` heu de crear la interfície `MedicamentService` amb el codi que es mostra a continuació:

```
1 package cat.xtec.ioc.service;
2
3 import cat.xtec.ioc.domain.Medicament;
4 import java.util.List;
5
6 public interface MedicamentService {
7
8     List<Medicament> getAllMedicaments();
9
10    Medicament getMedicamentById(String medicamentID);
11 }
```

L'objectiu és mostrar la llista de medicaments amb les crides ortodoxes (presentació a servei i aquest a persistència), i per això definiu `getAllMedicaments`. També aprofiteu per endreçar i afegiu el mètode `getMedicamentById`, que fareu servir quan vulgueu un medicament a partir del seu codi.

Al paquet `cat.xtec.ioc.service.impl` heu de crear la classe `MedicamentServiceImpl` que implementi la interfície anterior amb el codi que es mostra.

```
1 package cat.xtec.ioc.service.impl;
2
3 import cat.xtec.ioc.domain.Medicament;
4 import cat.xtec.ioc.domain.repository.MedicamentRepository;
5 import cat.xtec.ioc.service.MedicamentService;
6 import java.util.List;
7 import org.springframework.beans.factory.annotation.Autowired;
```

```
8 import org.springframework.stereotype.Service;
9
10 @Service
11 public class MedicamentServiceImpl implements MedicamentService {
12
13     @Autowired
14     private MedicamentRepository medicamentRepository;
15
16     public List<Medicament> getAllMedicaments() {
17         return medicamentRepository.getAllMedicaments();
18     }
19
20     public Medicament getMedicamentById(String medicamentID) {
21         return medicamentRepository.getMedicamentById(medicamentID);
22     }
23 }
```

En aquest cas, el servei no afegeix més regles de negoci, simplement es limita a cridar la persistència per obtenir el resultat. No obstant això, s'ha d'implementar aquesta capa perquè en el futur és possible que sorgeixin necessitats a implementar en aquesta capa. Per exemple, podríem dir que encara que obtingueu un medicament el retornareu només si l'usuari està autoritzat a la seva categoria.

3.1.2 Ordenant les crides des del controlador

Canvieu tot el controlador `MedicamentController` amb el codi que es mostra a continuació:

```
1 package cat.xtec.ioc.controller;
2
3 import cat.xtec.ioc.service.MedicamentService;
4 import java.io.IOException;
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.stereotype.Controller;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.servlet.ModelAndView;
12
13 @Controller
14 @RequestMapping("/medicaments")
15 public class MedicamentController {
16
17     @Autowired
18     private MedicamentService medicamentService;
19
20     @RequestMapping("/all")
21     public ModelAndView allMedicaments(HttpServletRequest request,
22                                       HttpServletResponse response)
23         throws ServletException, IOException {
24         ModelAndView modelview = new ModelAndView("medicaments");
25         modelview.getModelMap().addAttribute("medicaments", medicamentService.
26             getAllMedicaments());
27         return modelview;
28     }
29 }
```

Heu canviat la injecció del *repository* per la del nou servei. Ara sou a prop de l'ortodòxia a Spring MVC.

L' anotació `@RequestMapping` permet al Servlet Dispatcher encaminar la petició sobre el controlador específic que ha de processar-la.

Ara teniu dos `@RequestMapping`: un a nivell de classe amb l'URL `/medicaments` i un altre a nivell del mètode `allMedicaments` amb l'URL `/all`.

Si convenim que l'URL inicial de l'aplicació és *urlbase* (p. ex.: *localhost/stmedioc301*), quan el Dispatcher Servlet trobi l'URL *urlbase/medicaments/all*, amb */medicaments* determinarà que el controlador que ha de gestionar la petició és `MedicamentController`, i amb */all* el mètode a executar serà `allMedicaments`.

D'aquesta manera, dins de `MedicamentController` podríeu afegir més mètodes que agafin el control quan es doni una petició *urlbase/medicaments/xxx*, on *xxx* és l'URL que posaríeu a l' anotació `@RequestMapping` a nivell del mètode corresponent.

Proveu a executar l'aplicació afegint */medicaments/all* a l'URL, i us ha de mostrar la llista de medicaments.

3.2 Estoc de medicaments, medicaments per malaltia

Voleu que l'aplicació Estoc de medicaments mostri la llista de medicaments d'una categoria. Pel que sabeu, podeu fer els següents passos d'implementació:

- Un mètode del repositori que doni aquesta llista de medicaments a partir de la categoria donada com a paràmetre.
- Un mètode al servei que cridi aquest mètode del repositori.
- Un mètode al controlador que caci l'URL de la categoria i cridi el servei.

Però amb això, l' anotació `@RequestMapping` al controlador seria fixa amb la categoria com a constant, com per exemple *.../medicaments/Analgèsic*. Amb dues categories, com és el cas de la vostra aplicació, encara es pot suportar, però si en teniu algunes més ja no tindria sentit, ja que hauríeu de posar un `@RequestMapping` amb cada categoria com a URL. Encara més, no sabeu quines categories es poden crear en el futur, i per en cada una hauríeu de modificar el codi.

No us heu de preocupar, perquè podeu fer servir la funcionalitat de Spring MVC anomenada **plantilla d'URI** (*URI template pattern*).

En el cas d'Estoc de medicaments, teniu:

- *localhost:8080/stmedioc301/medicaments/Analgèsic*
- *localhost:8080/stmedioc301/medicaments/Anti-inflamatori*

on podríeu dir que la darrera part de l'URL és variable.

A Spring MVC, aquesta variable es nota entre claus i és una variable de ruta (*path variable*). Si l'anomenem categoria, llavors l'URL la podríeu considerar com *localhost:8080/stmedioc301/medicaments/{categoria}*.

Amb aquesta definició ja podeu començar a implementar la solució per mostrar els **medicaments per malaltia** amb una variable de ruta.

El codi del projecte "stmedioc" en l'estat **medicaments per malaltia** es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament és millor partir del que ja heu fet servir i copiar-lo amb el nom de "stmedioc301".

3.2.1 Adaptant repositori i servei

A la interfície `MedicamentRepository`, afegiu la següent declaració de mètode:

```
1 List<Medicament> getMedicamentsByCategory(String category);
```

I a la classe `InMemoryMedicamentRepository`, implementeu el mètode amb el codi que es mostra:

```
1 public List<Medicament> getMedicamentsByCategory(String category) {  
2     List<Medicament> medicamentsByCategory = new ArrayList<Medicament>();  
3     for (Medicament medicament : listOfMedicaments) {  
4         if (category.equalsIgnoreCase(medicament.getCategory())) {  
5             medicamentsByCategory.add(medicament);  
6         }  
7     }  
8     return medicamentsByCategory;  
9 }
```

El mètode `getMedicamentsByCategory` retornarà una *List* amb objectes `Medicament` de la categoria passada per paràmetre. Fixeu-vos que ignora diferències entre majúscules i minúscules.

A la interfície `MedicamentService`, afegiu la següent declaració de mètode:

```
1 List<Medicament> getMedicamentsByCategory(String category);
```

I a la classe `MedicamentServiceImpl`, implementeu el mètode amb el codi que es mostra:

```
1 public List<Medicament> getMedicamentsByCategory(String category) {  
2     return medicamentRepository.getMedicamentsByCategory(category);  
3 }
```

Simplement, es crida el repositori sense més regles de negoci a afegir.

3.2.2 Adaptant el controlador

A `MedicamentController`, afegireu l'anotació `@RequestMapping` per caçar qualsevol categoria, és a dir, fareu servir una variable de ruta (*path variable*). I al

mètode que segueix, recuperareu aquesta variable per passar-la com a paràmetre en la cerca de medicaments per categoria.

A la classe `MedicamentController`, afegiu l' anotació i el mètode amb el codi que es mostra:

```
1 @RequestMapping("/{category}")
2     public ModelAndView getMedicamentsByCategory(@PathVariable("category")
3         String medicamentCategory, HttpServletRequest request,
4         HttpServletResponse response)
5         throws ServletException, IOException {
6         ModelAndView modelAndView = new ModelAndView("medicaments");
7         modelAndView.getModelMap().addAttribute("medicaments", medicamentService.
            getMedicamentsByCategory(medicamentCategory));
            return modelAndView;
        }
```

Us demanarà afegir l'import següent:

```
1 import org.springframework.web.bind.annotation.PathVariable;
```

A `@RequestMapping` feu servir l' anotació `{pathVariable}` per indicar a Spring que l'URL contindrà un valor variable en aquesta part. En el vostre cas, el nom de la variable de ruta és *category*.

El mètode que es llança és `getMedicamentsByCategory`, i en els seus paràmetres heu fet servir l' anotació `@PathVariable` amb el format:

```
1 @PathVariable(pathVariable) type variableMetode
```

on `pathVariable` és el nom de la variable de ruta definida a l' anotació `@RequestMapping`, `variableMetode` és la variable que fareu servir al mètode, i `type` és el seu tipus. Així, Spring guarda el valor que s'ha posat a l'URL a `variableMetode` i ja la podeu fer servir amb aquest nom en el codi inclòs en el mètode.

`@PathVariable` sense valors de paràmetre implica que `pathVariable` té el mateix nom que `variableMetode`.

En el vostre cas, si executeu l'aplicació per exemple amb l'URL *localhost:8080/stmedioc302/medicaments/Analgèsic*, el `DispatcherServlet`, com que l'URL és *urlbase/medicaments/...*, passarà el control a `MedicamentController`. Dins d'aquest controlador es crida el mètode precedit per l' anotació `@RequestMapping("/{category}")`, ja que s'ha passat un URL del tipus *urlbase/medicaments/xxx*, on *xxx* és qualsevol valor diferent d'*all* (existeix `@RequestMapping("/{all}")`, que cridaria `allMedicaments`).

El mètode `getMedicamentsByCategory` cridat demana al repositori els medicaments de la categoria que s'ha passat per paràmetre i mostra la vista de medicaments només amb els que ha trobat.

Així, el resultat per a l'URL passat hauria de ser el que es mostra en la figura 3.1.

FIGURA 3.1. Sortida de medicaments per categoria

Medicaments	
Llista de medicaments en magatzem	
Paracetamol Paracetamol 1 g 2,6 € 56 unitats en magatzem	Àcid acetilsalicílic Àcid acetilsalicílic 2,6 € 15 unitats en magatzem

3.3 "Estoc de medicaments", llista de medicaments segons filtre

“Estoc de medicaments” pot mostrar medicaments d’una categoria passant l’URL amb `/medicaments/nomDeCategoria`, on `nomDeCategoria` serà rebut al controlador específic amb l’annotació `@RequestMapping("/{category}")` que conté la variable de ruta (*path variable*).

Els vostres requeriments es veuen ampliats amb la necessitat de mostrar medicaments segons filtres. Enriqueixeu l’aplicació acceptant URL del tipus `localhost:8080/stmedioc303/medicaments/filter/ByCriteria;producer=Ferrer,Bayer;estoc=1,100`, que, per exemple, podeu traduir com a llista de medicaments dels productors Ferrer o Bayer amb unitats en estoc entre 1 i 100. Però com veureu, la traducció és un conveni, perquè depèn del que implementeu a la consulta que fareu a les dades.

En realitat, el que voleu implementar és l’acceptació de variables de matriu (*matrix variables*) com a paràmetres i com fer-les servir al vostre codi.

Aquestes variables es convertiran en un `Map<String,List<String>>` a partir de l’annotació `@MatrixVariable`. Fixeu-vos que té sentit, perquè `producer=Ferrer,Bayer;estoc=1,100` es pot representar com una *Map* amb les *keys* `producer` i `estoc`, i els valors són llistes.

Implementeu l’exemple que dona resposta a `localhost:8080/stmedioc303/medicaments/filter/ByCriteria;producer=Ferrer,Bayer;estoc=1,100` amb el significat que hem convingut.

3.3.1 Adaptant repositori i servei

A la interfície `MedicamentRepository`, afegiu la següent declaració de mètode:

```
1 Set<Medicament> getMedicamentsByFilter(Map<String, List<String>> filterParams);
```

I a la classe `InMemoryMedicamentRepository`, implementeu el mètode amb el codi que es mostra.

```
1 public Set<Medicament> getMedicamentsByFilter(Map<String, List<String>>
    filterParams) {
```

El codi del projecte “stmedioc” en l’estat **llista de medicaments segons filtre** es pot descarregar des de l’enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament és millor partir del que ja heu fet servir i copiar-lo amb el nom de “stmedioc303”.

```

2      Set<Medicament> medicamentsByProducer = new HashSet<Medicament>();
3      Set<Medicament> medicamentsInStockRange = new HashSet<Medicament>();
4      Set<String> criterias = filterParams.keySet();
5      long minStock = 0;
6      long maxStock = 0;
7      if (criterias.contains("producer")) {
8          for (String producerName : filterParams.get("producer")) {
9              for (Medicament medicament : listOfMedicaments) {
10                 if (producerName.equalsIgnoreCase(medicament.getProducer())
11                    ) {
12                     medicamentsByProducer.add(medicament);
13                 }
14             }
15         }
16         if (criterias.contains("estoc")) {
17             minStock = Long.parseLong(filterParams.get("estoc").get(0));
18             maxStock = Long.parseLong(filterParams.get("estoc").get(1));
19
20             for (Medicament medicament : listOfMedicaments) {
21                 if ((medicament.getStockQuantity() >= minStock) && (medicament.
22                     getStockQuantity() <= maxStock)) {
23                     medicamentsInStockRange.add(medicament);
24                 }
25             }
26         }
27         medicamentsInStockRange.retainAll(medicamentsByProducer);
28         return medicamentsInStockRange;
29     }

```

Set és una interfície del framework Java Collections. Es pot consultar la referència a la mateixa documentació d'Oracle a bit.ly/2r3n02Y.

Es defineixen dues col·leccions (`medicamentsByProducer` i `medicamentsInStockRange`) que contindran els medicaments que compleixen els criteris per retornar només una amb la intersecció d'elements.

Es defineix un *Set* de criteris que en el vostre cas contindrà *producer* i *estoc*. Fixeu-vos que podreu provar sense algun dels criteris, i llavors aquest no es tindrà en compte.

En el cas del criteri *producer*, se seleccionen els elements del paràmetre amb aquest criteri, es recorre aquesta selecció i per a cada valor s'afegeixen els medicaments amb aquest *producer* al *Set* `medicamentsByProducer`.

En el cas del criteri *estoc*, es fan servir les variables *minStock* i *maxStock* per assignar respectivament els elements 0 i 1 de la llista de paràmetres amb la *key* *estoc* (`filterParams.get("estoc").get(i)`). Llavors, es van afegint sobre el *Set* `medicamentsInStockRange` els medicaments amb unitats en estoc en el rang *[minStock,maxStock]*.

Finalment, com que voleu retornar la llista de medicaments que compleixen tots els criteris a la vegada, agafeu una de les llistes i només la deixeu amb els medicaments que coincideixen amb l'altre, per finalment retornar la intersecció.

```
1 medicamentsInStockRange.retainAll(medicamentsByProducer);
```

Un cop heu definit la consulta a les dades, passeu a modificar la capa de servei.

A la interfície `MedicamentService`, afegiu la següent declaració de mètode:

```
1 Set<Medicament> getMedicamentsByFilter(Map<String, List<String>> filterParams);
```


I a la classe `MedicamentServiceImpl`, implementeu el mètode amb el codi que es mostra:

```
1 public Set<Medicament> getMedicamentsByFilter(Map<String, List<String>>
   filterParams) {
2     return medicamentRepository.getMedicamentsByFilter(filterParams);
3 }
```

Simplement es crida el repositori, sense més regles de negoci a afegir.

3.3.2 Adaptant el controlador

Al controlador específic voleu fer servir variables tipus matriu, com per exemple les que es mostren a l'URL `localhost:8080 /stmedic303/medicaments/filter/ByCriteria;producer=Ferrer,Bayer;estoc=1,100`.

Tal com heu implementat al mètode `getMedicamentsByFilter` del repositori, heu convingut que això voldrà dir el conjunt de medicaments amb productor Bayer o Ferrer, i a més l'estoc dels quals estigui entre 1 i 100 unitats.

Per aconseguir això es farà servir l'anotació `@MatrixVariable`, però, a més, a Spring MVC es necessita una configuració addicional.

A `DispatcherServlet-servlet.xml` canvieu l'etiqueta `mvc:annotation-driven` així:

```
1 <mvc:annotation-driven enable-matrix-variables="true"/>
```

A la classe `MedicamentController`, afegiu l'anotació i el mètode amb el codi que es mostra:

```
1 @RequestMapping("/filter/{ByCriteria}")
2 public ModelAndView getMedicamentsByFilter(@MatrixVariable(pathVar = "
   ByCriteria") Map<String, List<String>> filterParams,
   HttpServletRequest request, HttpServletResponse response)
3     throws ServletException, IOException {
4     ModelAndView modelAndView = new ModelAndView("medicaments");
5     modelAndView.getModelMap().addAttribute("medicaments",
   medicamentService.getMedicamentsByFilter(filterParams));
6     return modelAndView;
7 }
```

A `@RequestMapping` feu servir l'anotació `{pathVariable}` per indicar a Spring que l'URL contindrà un valor variable en aquesta part. En el vostre cas, el nom de la variable és `ByCriteria`.

El mètode que es llança és `getMedicamentsByFilter`, i en els seus paràmetres hem fet servir l'anotació `@MatrixVariable` amb el format:

```
1 @MatrixVariable(pathVar = "pathVariable") Map<String, List<String>>
   filterParams
```

on `pathVariable` és el nom de la variable definida a l'anotació `@RequestMapping` i `filterParams` és la variable que farem servir al mètode amb tipus `Map`. Així, Spring guarda el valor que s'ha posat a l'URL a `filterParams` i ja la podeu fer servir amb aquest nom en el codi inclòs en el mètode.

L'anotació `@RequestMapping` permet definir més d'un grup de criteris, és a dir, més d'un grup de *matrix variables*. El format seria així: `@RequestMapping("/filter/{ByCriteria}/{ByOtherCriteria}")`.

En el vostre cas, si executeu l'aplicació per exemple amb l'URL `localhost:8080/stmedioc303/medicaments/filter/ByCriteria;producer=Ferrer,Bayer;estoc=1,100`, com que l'URL és `urlbase/medicaments/...`, el `DispatcherServlet` passarà el control a `MedicamentController`. Dins d'aquest controlador es crida el mètode precedit per l'anotació `@RequestMapping("/filter/{ByCriteria}")`, ja que s'ha passat un URL del tipus `urlbase/medicaments/filter/ByCriteria;xxx`, on `xxx` és un conjunt de parells clau=llista de valors.

El mètode `getMedicamentsByFilter` cridat demana al repositori els medicaments amb el filtre que s'ha passat per paràmetre i mostra la vista `medicaments` només amb els que ha trobat. Fixeu-vos que el paràmetre “filtre”, conjunt de parells clau=llista de valors, s'ha convertit a un `Map<String, List<String>>` que anomenem `filterParams`, i amb aquest nom el fem servir al codi del mètode.

Així, el resultat per a l'URL passat hauria de ser el que es mostra en la figura 3.2.

FIGURA 3.2. Sortida de la llista de medicaments amb filtre

Medicaments	
Llista de medicaments en magatzem	
Paracetamol Paracetamol 1 g 2,6 € 56 unitats en magatzem	Àcid acetilsalicílic Àcid acetilsalicílic 2,6 € 15 unitats en magatzem

Podeu provar URL amb altres rangs d'estoc i altres productors.

3.4 "Estoc de medicaments", detall de medicament

“Estoc de medicaments” mostra llistes de medicaments que compleixen certs criteris basats en variables que posem a l'URL. És a dir, en els casos que es gestionen amb les anotacions `@PathVariable` i `@MatrixVariable`, s'estan rebent els valors com a part de l'URL i no com a variables que formen part d'*HTTP request*.

Amplieu l'aplicació per tal d'acceptar variables a la petició *HTTP* (*HTTP request*). Per exemple, analitzant l'URL `localhost:8080/stmedioc304/medicaments/medicament?codi=M020` veieu que `codi` serà passat al servidor dins del cos de la petició *HTTP* com una variable.

Aquests tipus d'URL tenen el format `urlbase/...?var1=valor1&var2=valor2&...&varn=valorn`, on el símbol “?” indica que comença la llista de variables

i el símbol & separa cada variable. Cada parell `variable=valor` serà passat al servidor web dins del cos de l'*HTTP request* (petició *HTTP*).

Per exemple, encara que doni un error, si obriu el navegador i Firebug i proveu l'URL `localhost/?var1=valor1&var2=valor2&var3=valor3`, veureu que heu fet un get i ha passat per paràmetre:

- var1 valor1
- var2 valor2
- var3 valor3

I aquest fet no es dona quan passeu les URL del tipus que gestionen `@PathVariable` i `@MatrixVariable`, ja que no es correspon amb la sintaxi `urlbase/...?var1=valor1&var2=valor2&...&varn=valorn` i, per tant, no hi ha paràmetres al cos de la petició HTTP.

A “Estoc de medicaments”, per il·lustrar el pas de variables a la petició *HTTP*, veureu com mostrar el detall d'un medicament a partir d'un URL del tipus esmentat, és a dir, passant els valors com a variables del cos de l'*HTTP request*. Per fer això fareu servir l'anotació `@RequestParam`.

No només veureu això, també fareu que des de qualsevol llista de medicaments, filtrada o no, es pugui anar al detall de qualsevol d'ells amb un botó, i així fareu servir el que heu preparat per mostrar el detall.

3.4.1 Mostrant el detall

Heu de mostrar el detall de medicament en una nova vista que anomenareu `medicament.jsp`. Cal aquesta vista, perquè a les llistes no es mostren totes les dades del medicament, com per exemple la categoria.

Aquesta nova vista es mostrarà quan es doni una petició del tipus `localhost:8080/stmedioc304/medicaments/medicament?codi=M020`.

I per tant, a `MedicamentController` haureu d'afegir l'anotació que agafi el control i el mètode a disparar.

A les diferents capes d'Estoc de medicaments ja teniu la possibilitat d'obtenir un objecte `Medicament` a partir del seu codi, i per això us estalvieu codificar en altres capes, a banda de la de presentació.

A la classe `MedicamentController`, afegiu l'anotació i mètode amb el codi que es mostra;

El codi del projecte “stmedioc” en l'estat detall de medicament es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament és millor partir del que ja heu fet servir i copiar-lo amb el nom de “stmedioc304”.

```
1 @RequestMapping("/medicament")
2 public ModelAndView getMedicamentById(@RequestParam("codi") String
    medicamentId, HttpServletRequest request, HttpServletResponse response
    )
```

```

3         throws ServletException, IOException {
4             ModelAndView modelAndView = new ModelAndView("medicament");
5             modelAndView.getModelMap().addAttribute("medicament", medicamentService.
6                 getMedicamentById(medicamentId));
7             return modelAndView;
            }

```

L'annotació `@RequestParam` permet assignar la variable passada per paràmetre des de la petició, amb la variable que es farà servir en el mètode.

En el nostre cas, l'annotació `@RequestParam` enllaça el paràmetre codi del cos de la petició amb la variable `medicamentId` que fareu servir dins del mètode.

El mètode crida `getMedicamentById` de la capa de servei que ja està implementat. Aquest retorna l'objecte `Medicament` amb el codi que s'ha passat pel paràmetre.

A la carpeta `views`, afegiu la nova vista `medicament.jsp` amb el codi que es mostra a continuació:

```

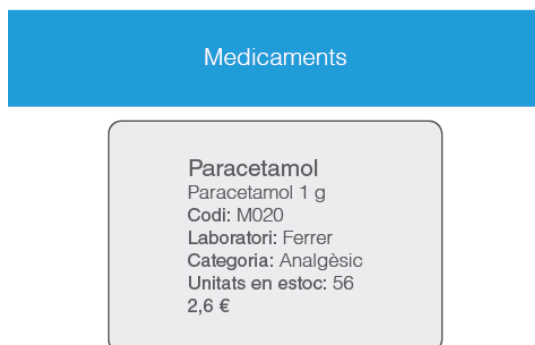
1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@page contentType="text/html" pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5     <head>
6         <meta http-equiv="Content-Type" content="text/html; charset=ISO
7             -8859-1">
8         <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/
9             css/bootstrap.min.css">
10        <title>Medicament</title>
11    </head>
12    <body>
13        <section>
14            <div class="jumbotron">
15                <div class="container">
16                    <h1>Medicament</h1>
17                </div>
18            </div>
19        </section>
20        <section class="container">
21            <div class="row">
22                <div class="col-md-5">
23                    <h3>${medicament.name}</h3>
24                    <p>${medicament.description}</p>
25                    <p>
26                        <strong>Codi : </strong>${medicament.medicamentId}
27                    </p>
28                    <p>
29                        <strong>Laboratori</strong> : ${medicament.producer}
30                    </p>
31                    <p>
32                        <strong>Categoria</strong> : ${medicament.category}
33                    </p>
34                    <p>
35                        <strong>Unitats en estoc </strong> :
36                        ${medicament.stockQuantity}
37                    </p>
38                    <h4>${medicament.price} USD</h4>
39                </div>
40            </div>
41        </section>
42    </body>
43 </html>

```

La vista mostrarà els valors de l'objecte de nom `medicament` que havíeu afegit com a atribut al controlador.

Si executeu l'aplicació al navegador amb l'URL `localhost:8080/stmedioc304/medicaments/medicament?codi=M020`, el resultat és el que es veu en la figura 3.3.

FIGURA 3.3. Sortida de detall de medicament



La part de l'URL `/medicaments` ha fet que el `Dispatcher Servlet` passi la responsabilitat a `MedicamentController`. L'URL és `/medicaments/medicament`, i com que existeix l'anotació `@RequestMapping("/medicament")` dispararà el mètode `getMedicamentById` que la segueix.

Aquest mètode té anotacions del tipus `@RequestParam`, i per tant farà servir els paràmetres del cos de la petició. En concret, l'URL conté `?codi=M020`, que vol dir que caçarà el paràmetre `codi` a partir de `@RequestParam("codi") String medicamentId` i l'assignarà a `medicamentId`, que després es fa servir per cercar l'objecte `Medicament` amb igual `codi`. Aquest objecte `Medicament` és ficat en un atribut `medicament` a la resposta (*response*).

Llavors, la vista `medicament.jsp` mostra les dades d'aquest medicament fent servir l'atribut esmentat. Proveu amb altres codis de medicaments per veure'n el funcionament.

3.4.2 Arribant al detall des de la llista

"Estoc de medicaments" sap com mostrar el detall d'un medicament a partir de paràmetres de petició, és a dir, amb URL del tipus `localhost:8080/stmedioc304/medicaments/medicament?codi=xxx`, on `xxx` és el `medicamentId`.

Aprofiteu el que heu fet per mostrar el detall de qualsevol medicament a partir de la llista. Afegireu un botó a cada element de la llista que ens porti a la vista de detall. I per no anar canviant URL, fareu que des del detall es pugui tornar a la llista.

A `medicaments.jsp`, afegiu la directiva:

```
1 <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
```

i just després de:

```
1 <p>Hi ha ${medicament.stockQuantity} unitats en magatzem</p>
```

afegiu el codi següent:

```
1 <p>
2
3         <a href=" <spring:url value= "/medicaments/
4             medicament?codi=${medicament.
5             medicamentId}" /> " class="btn btn-
            primary">
            <span class="glyphicon-info-sign
              glyphicon"/></span> Detall
        </a>
</p>
```

L'enllaç té l'etiqueta `<spring:url>`, que construirà un URL Spring MVC vàlida. En aquest cas, el codi que farà servir l'obté de l'objecte `medicament` de la llista que s'està recorrent.

Executeu l'aplicació amb l'URL `localhost:8080/stmedioc304/medicaments/all` i obtindreu un resultat com el que mostra figura 3.4.

FIGURA 3.4. Sortida de llista de medicaments amb botó de detall

Medicaments		
Llista de medicaments en magatzem		
Ibuprofèn Ibuprofèn 600 mg 2,0 € 214 unitats en magatzem i Detall	Paracetamol Paracetamol 1 g 2,6 € 56 unitats en magatzem i Detall	Àcid acetilsalicílic Àcid acetilsalicílic 2,6 € 15 unitats en magatzem i Detall

Si premeu al detall del Paracetamol, l'aplicació ha de mostrar el detall (vegeu la figura 3.5).

FIGURA 3.5. Sortida de detall de medicament

Medicaments
Paracetamol Paracetamol 1 g Codi: M020 Laboratori: Ferrer Categoria: Analgèsic Unitats en estoc: 56 2,6 €

Ara afegireu al detall un botó per tornar a la llista. Per fer això també es farà servir l'etiqueta `<spring:url>`.

A `medicament.jsp`, afegiu la directiva:

```
1 <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
```

i just després de

```
1 <h4>${medicament.price} USD</h4>
```

afegiu el codi següent:

```
1 <a href="<spring:url value="/medicaments/all" />" class="btn btn-default">
2     <span class="glyphicon-hand-left glyphicon"></span>
3         tornar
    </a>
```

Si executeu l'aplicació i aneu a mostrar la llista de medicaments ja podeu prémer un botó de detall, que farà que es mostri la vista de detall, i després tornar una altra vegada a la llista.

3.5 Què s'ha après?

`MedicamentController` crida directament a la capa de servei `MedicamentService` i no directament a la persistència.

L'anotació `@RequestMapping("/medicaments")` a nivell de classe, fa que qualsevol URL del tipus `/medicaments` agafi aquest controlador. Les anotacions `@RequestMapping("/xxx")` posades al davant de diversos mètodes, fa que aquests es disparin quan es fan peticions amb URL del tipus `/medicaments/xxx`.

Peticions com `localhost:8080/stmedioc301/medicaments/Analgèsic`, on “analgèsic” es pot substituir per qualsevol categoria, són ateses pel mètode que segueix a l'anotació:

```
1 @RequestMapping("/{category}")
```

on `category` és la variable de ruta (*variable path*) que és enllaçada a la variable `medicamentCategory` amb l'anotació:

```
1 @PathVariable("category") String medicamentCategory.
```

Les variables de matriu (*matrix variables*), permeten llegir URL del tipus `localhost:8080/stmedioc303/medicaments/filter/ByCriteria;producer=Ferrer,Bayer;es toc=1,100`, on el conjunt de parell clau=llista de valors és la matriu.

Amb l'anotació:

```
1 @RequestMapping("/filter/{ByCriteria}")
```

es passa el control al mètode que la segueix, i en aquest, l'anotació:

```
1 @MatrixVariable(pathVar = "ByCriteria") Map<String, List<String>> filterParams
```

assigna el paràmetre (la matriu) al *Map* `filterParams`, variable que es pot fer servir dins del mètode.

Poden venir paràmetres al cos d'una petició HTTP, per exemple, a una petició `Post` d'un formulari.

En general, es pot atendre a peticions amb URL del tipus *urlbase/...?var1=valor1&var2=valor2&...&varn=valorn*, on el que segueix el símbol `?` és el conjunt de paràmetres que aniran en el cos de la petició HTTP.

Per exemple, es pot resoldre una URL com:

localhost:8080/stmedioc304/medicaments/medicament?codi=M020

amb l' anotació:

```
1 @RequestMapping("/medicament")
```

precedint al mètode i com a paràmetre d'aquest, l'anotació:

```
1 @RequestParam("codi") String medicamentoId
```

que enllaça el paràmetre amb la variable `medicamentoId` que es pot fer servir dins del mètode.

L'etiqueta de Spring `<spring:url>` serveix per enllaçar una pàgina amb una altra construint un URL vàlid.

4. Spring MVC, altres aplicacions

Partim de l'aplicació web Estoc de medicaments, que mostra una pàgina de benvinguda i, mitjançant l'URL adient, pot proporcionar la llista de tots els medicaments, els d'una categoria o els medicaments que compleixen cert filtre.

També pot mostrar el detall d'un medicament directament amb un URL o des de la llista de medicaments. Afegit a tot això, l'aplicació pot actualitzar l'estoc d'un medicament.

Aquesta aplicació segueix l'arquitectura i el procés de qualsevol aplicació Spring MVC. S'estructura en les quatre capes següents:

- presentació (*Presentation*)
- domini (*Domain*)
- servei (*Service*)
- persistència (*Persistence*)

Estoc de medicaments fa servir anotacions de Spring MVC:

- `@Controller`, `@Repository` i `@Service`, que defineixen el tipus d'objecte Spring.
- `@RequestMapping`, que caça l'URL definit i executa el mètode que el segueix.
- `@Autowired`, que injecta l'objecte que segueix.
- `@PathVariable` i `@MatrixVariable` per accedir a variables dins de l'URL.
- `@RequestParam` per enllaçar amb els paràmetres del cos de la petició (`Request params`).

A més, Estoc de medicaments conté els fitxers XML amb la configuració necessària per dirigir el comportament de Spring MVC respecte a l'aplicació.

Anem a afegir funcionalitat a l'aplicació Estoc de medicaments que necessitem en la majoria d'aplicacions web.

Afegirem a Estoc de medicaments un formulari per crear medicaments, la qual cosa us permetrà veure totes les possibilitats que ofereix Spring MVC per tractar els formularis, sobretot com passar dades des de la vista fins al model, perquè fins ara el sentit havia estat al contrari.

Construireu una pàgina de *login* (identificació) i el codi necessari a la resta de capes. Al voltant d'aquest objectiu, veureu etiquetes Spring relacionades amb la seguretat i diverses possibilitats de configuració.

Fareu servir la **internacionalització** de Spring per mostrar com fer l'aplicació multiidioma (parcialment), i veureu què són i com es comporten els **interceptors** de Spring.

Fareu servir diverses etiquetes pròpies de Spring amb les directives:

```
1 <%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
2 <%@taglib prefix="spring" uri="http://www.springframework.org/tags" %>
```

Aquestes llibreries d'etiquetes proporcionen molta funcionalitat per als formularis, com ara la validació d'errors, però també per a altres finalitats.

4.1 Estoc de medicaments, formulari per afegir medicaments

A Estoc de medicaments, a tota la funcionalitat que hi vegeu implementada, el flux de dades és des del model fins a la vista.

Per veure com es genera i manega a Spring MVC un flux de dades des de la vista fins al model afegireu un formulari l'acció del qual sigui crear un medicament.

Un *form-backing bean* de Spring MVC és l'objecte que permet associar les dades d'un formulari web amb l'aplicació que rep la petició.

L'anotació `@ModelAttribute` i l'objecte *form-backing bean* ens permetran crear el medicament a partir de les dades del formulari.

Els paràgrafs següents descriuen el procés que heu de seguir per implementar el formulari.

Per presentar el formulari implementareu un primer mètode que passarà un objecte `Medicament` buit (`newMedicament`) com atribut a la nova vista `addMedicament`.

A la vista, l'etiqueta de formulari serà de la llibreria de Spring i associarà bidireccionalment `newMedicament` al formulari, és a dir, cada camp del formulari estarà també associat a les propietats de `newMedicament` i l'actualització d'uns farà que s'actualitzin els altres. Per això es diu que els objectes com `newMedicament` són els *form-backing bean*, ja que és on Spring MVC guarda els valors del formulari en el context.

Finalment, implementareu un segon mètode que reculli el *submit* del formulari. L'anotació:

```
1 @ModelAttribute("newMedicament") Medicament newMedicamentToAdd
```

us permetrà fer servir els valors del formulari via `newMedicamentToAdd`.

4.1.1 Treballant les capes de domini, persistència i servei

A la classe `Medicament`, afegiu un constructor sense paràmetres.

```
1 public Medicament() {  
2     }
```

És necessari, perquè necessitem crear un objecte `medicament` buit.

A la interfície `MedicamentRepository`, afegiu la següent declaració de mètode:

```
1 void addMedicament(Medicament medicament);
```

I a la classe `InMemoryMedicamentRepository`, implementeu el mètode amb el codi que es mostra.

```
1 public void addMedicament(Medicament medicament) {  
2     listOfMedicaments.add(medicament);  
3 }
```

Veieu que simplement s'afegeix un objecte `Medicament` a la vostra llista de medicaments en memòria. Això és equivalent a un INSERT en el cas que la nostra persistència hagués estat implementada amb una base de dades (relacional amb SQL).

A la interfície `MedicamentService`, afegiu la següent declaració de mètode:

```
1 void addMedicament(Medicament medicament);
```

I a la classe `MedicamentServiceImpl`, implementeu el mètode amb el codi que es mostra.

```
1 public void addMedicament(Medicament medicament) {  
2     medicamentRepository.addMedicament(medicament);  
3 }
```

Simplement es crida el repositori sense més regles de negoci a afegir.

4.1.2 Formulari a la capa de presentació

Al controlador heu d'implementar els mètodes inclosos en el procés que havíeu de seguir. És a dir, un mètode per mostrar el formulari i un mètode per processar-lo.

A la classe `MedicamentController`, heu d'afegir els mètodes amb el codi que es mostra.

```
1 @RequestMapping(value = "/add", method = RequestMethod.GET)  
2     public ModelAndView getAddNewMedicamentForm(HttpServletRequest request,  
3         HttpServletResponse response)  
4         throws ServletException, IOException {
```

```

4      ModelAndView modelAndView = new ModelAndView("addMedicament");
5      Medicament newMedicament = new Medicament();
6      modelAndView.getModelMap().addAttribute("newMedicament", newMedicament);
7      return modelAndView;
8  }
9
10     @RequestMapping(value = "/add", method = RequestMethod.POST)
11     public String processAddNewMedicamentForm(@ModelAttribute("newMedicament")
12         Medicament newMedicamentToAdd) {
13         medicamentService.addMedicament(newMedicamentToAdd);
14         return "redirect:/medicaments/all";
15     }

```

El mètode `getAddNewMedicamentForm` és el que fareu servir per mostrar el formulari. En aquest cas, es dispararà amb una petició GET (`method = RequestMethod.GET`) creant un objecte buit `Medicament` que retorna a la vista `addMedicament` com a atribut de nom `newMedicament`. L'associació que fareu en aquesta vista el converteix en el *form-backing bean*.

El mètode `processAddNewMedicamentForm` és el que fareu servir per processar el formulari. En aquest cas, es dispararà amb una petició POST (`method = RequestMethod.POST`). L'anotació:

```

1  @ModelAttribute("newMedicament") Medicament newMedicament

```

associa les dades del formulari (guardades a `newMedicament`) a la variable `newMedicamentToAdd` que fareu servir al codi del mètode.

Com que no heu de tornar un `ModelAndView` perquè només voleu redirigir a la vista que mostra la llista de medicaments, feu que `processAddNewMedicamentForm` sigui de tipus *string*. No manegueu `ModelAndView` ni necessiteu `HttpServletRequest request` i `HttpServletResponse response` com a paràmetres.

La instrucció:

```

1  return "redirect:/medicaments/all";

```

fa que el navegador faci una nova petició amb aquest URL i, per això, el `Dispatcher Servlet` tornarà a cercar quin controlador específic s'encarrega de l'URL passat com a *redirect*. En aquest cas és el mateix controlador, i mostrarà la llista de tots els medicaments.

Abans de codificar la vista, a l'arxiu de configuració `web.xml` heu d'afegir el filtre `CharacterEncodingFilter` amb les etiquetes que es mostren.

```

1  <filter>
2      <filter-name>encodingFilter</filter-name>
3      <filter-class>org.springframework.web.filter.CharacterEncodingFilter</
4          filter-class>
5      <init-param>
6          <param-name>encoding</param-name>
7          <param-value>UTF-8</param-value>
8      </init-param>
9      <init-param>
10         <param-name>forceEncoding</param-name>
11         <param-value>true</param-value>

```

```
11     </init-param>
12 </filter>
13 <filter-mapping>
14     <filter-name>encodingFilter</filter-name>
15     <url-pattern>/*</url-pattern>
16 </filter-mapping>
```

Aquest filtre permet la utilització d'accents als camps del nostre formulari. En realitat, `CharacterEncodingFilter` permet l'especificació de la codificació de caràcters que forcem en les peticions, ja que actualment existeixen navegadors que no fan cas de la codificació especificada en la pàgina HTML (a la vista, en el vostre cas).

Afegiu a la carpeta *views* una nova vista `addMedicament.jsp` amb el codi que es mostra a continuació:

```
1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
3 <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
4 <%@page contentType="text/html" pageEncoding="UTF-8"%>
5 <!DOCTYPE html>
6 <html>
7     <head>
8         <meta http-equiv="Content-Type" content="text/html; charset=ISO
9             -8859-1">
10         <link
11             rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/
12             css/bootstrap.min.css">
13         <title>Medicaments</title>
14     </head>
15     <body>
16         <section>
17             <div class="jumbotron">
18                 <div class="container">
19                     <h1>Medicament</h1>
20                     <p>Afegir medicament</p>
21                 </div>
22             </div>
23             <section class="container">
24                 <form:form modelAttribute="newMedicament" class="form-horizontal">
25                     <fieldset>
26                         <legend>Afegir medicament</legend>
27                         <div class="form-group">
28                             <label class="control-label col-lg-2 col-lg-2" for="
29                                 medicamentId">Codi</label>
30                             <div class="col-lg-10">
31                                 <form:input id="medicamentId" path="medicamentId"
32                                     type="text" class="form:input-large"/>
33                             </div>
34                         </div>
35                         <div class="form-group">
36                             <label class="control-label col-lg-2 col-lg-2" for="
37                                 name">Nom</label>
38                             <div class="col-lg-10">
39                                 <form:input id="name" path="name" type="text" class
40                                     ="form:input-large"/>
41                             </div>
42                         </div>
43                         <div class="form-group">
44                             <label class="control-label col-lg-2 col-lg-2" for="
45                                 price">Preu</label>
46                             <div class="col-lg-10">
47                                 <form:input id="price" path="price" type="text"
48                                     class="form:input-large"/>
49                             </div>
50                         </div>
51                     </fieldset>
52                 </form:form>
53             </section>
54         </body>
55 </html>
```

```

43     </div>
44     <div class="form-group">
45         <label class="control-label col-lg-2 col-lg-2" for="
46             producer">Laboratori</label>
47         <div class="col-lg-10">
48             <form:input id="producer" path="producer" type="
49                 text" class="form:input-large"/>
50         </div>
51     </div>
52     <div class="form-group">
53         <label class="control-label col-lg-2 col-lg-2" for="
54             category">Categoria</label>
55         <div class="col-lg-10">
56             <form:input id="category" path="category" type="
57                 text" class="form:input-large"/>
58         </div>
59     </div>
60     <div class="form-group">
61         <label class="control-label col-lg-2 col-lg-2" for="
62             stockQuantity">Unitats en estoc</label>
63         <div class="col-lg-10">
64             <form:input id="stockQuantity" path="stockQuantity"
65                 type="text" class="form:input-large"/>
66         </div>
67     </div>
68     <div class="form-group">
69         <label class="control-label col-lg-2 col-lg-2" for="
70             stockInOrder">Unitats en comandes</label>
71         <div class="col-lg-10">
72             <form:input id="stockInOrder" path="stockInOrder"
73                 type="text" class="form:input-large"/>
74         </div>
75     </div>
76     <div class="form-group">
77         <label class="control-label col-lg-2"
78             for="description">Descripció</label>
79         <div class="col-lg-10">
80             <form:textarea id="description" path="description"
81                 rows = "2"/>
82         </div>
83     </div>
84     <div class="form-group">
85         <label class="control-label col-lg-2"
86             for="active">actiu</label>
87         <div class="col-lg-10">
88             <form:radio button path="active" value="true" />Si
89             <form:radio button path="active" value="false" />No
90         </div>
91     </div>
92     <div class="form-group">
93         <div class="col-lg-offset-2 col-lg-10">
94             <input type="submit" id="btnAdd" class="btn btn-
95                 primary"
96                 value = "Crear"/>
97         </div>
98     </div>
99 </fieldset>
100 </form:form>
101 </section>
102 </body>
103 </html>

```

Fixeu-vos que es fan servir les llibreries d'etiquetes de Spring que aporten més funcionalitat.

L'etiqueta que defineix el formulari:

```
1 <form:form modelAttribute="newMedicament" class="form-horizontal">
```

està associant els camps del formulari amb el *form-backing bean* `newMedicament`, de manera que quan s'envii el formulari, el controlador podrà agafar els valors d'aquest objecte.

Les etiquetes dels camps del formulari:

```
1 <form:input id="medicamentId" path="medicamentId" type="text" class="form:input  
  -large"/>
```

associen individualment cada control del formulari amb la propietat del *form-backing bean* mitjançant l'atribut `path`.

Si executeu l'aplicació amb l'URL `localhost:8080/stmedioc401/medicaments/add` us mostrarà la pàgina amb el formulari buit (vegeu la figura 4.1).

FIGURA 4.1. Sortida de l'aplicació amb el formulari buit

Medicament
Afegeix medicament

Afegeix medicament

Codi

Nom

Preu

Laboratori

Categoria

Unitats en estoc

Unitats en comanda

Descripció

Actiu Sí ☐ No ☒

Crea

Ompliu el formulari tal com es mostra en la figura 4.2.

FIGURA 4.2. Sortida de l'aplicació amb el formulari emplenat

Medicament
Afegeix medicament

Afegeix medicament

Codi

Nom

Preu

Laboratori

Categoria

Unitats en estoc

Unitats en comanda

Descripció

Actiu Sí ☒ No ☐

Crea

En prémer *Crear* us mostrarà la llista de medicaments amb el nou medicament afegit (vegeu la figura 4.3).

FIGURA 4.3. Sortida de la llista amb el medicament afegit

<div>Medicaments</div> <div>Llista de medicaments en magatzem</div>			
Ibuprofèn Ibuprofèn 600 mg 2,0 € 214 unitats en magatzem	Paracetamol Paracetamol 1 g 2,6 € 56 unitats en magatzem	Àcid acetilsalílic Àcid acetilsalílic 2,6 € 15 unitats en magatzem	Crema Cosmètica per a les mans 12,0 € 12 unitats al magatzem
i Detall	i Detall	i Detall	i Detall

4.2 Estoc de medicaments, llista blanca al formulari

El formulari per afegir medicaments de l'aplicació Estoc de medicaments conté totes les propietats de Medicament. S'hi inclouen tots els camps, i alguns no es poden omplir perquè és impossible tenir la informació en crear el medicament, com per exemple les unitats en comanda (encara no hem pogut fer cap comanda).

Spring MVC permet registrar una llista de propietats (*Whitelist*) que poden obviar-se en una petició d'un formulari.

Si una propietat de la *Whitelist* és a la petició, podem identificar-la i decidir què fer. Per exemple, llançar una excepció.

Això s'aconsegueix afegint al controlador específic l'anotació `@InitBinder` i la implementació del mètode que es dispararà. En aquest mètode es posarà la llista de propietats sota la restricció esmentada, és a dir, la *Whitelist*.

Modifiqueu `MedicamentController` afegint el mètode per identificar la *Whitelist* (en el vostre cas només és `stockInOrder`). El codi és el que es mostra a continuació:

```

1 @InitBinder
2   public void initialiseBinder(WebDataBinder binder) {
3       binder.setDisallowedFields("stockInOrder");
4   }

```

A la mateixa classe, canvieu el mètode `processAddNewMedicamentForm` pel codi següent.

```

1 public String processAddNewMedicamentForm(@ModelAttribute("newMedicament")
2     Medicament newMedicamentToAdd, BindingResult result) {
3     String[] suppressedFields = result.getSuppressedFields();
4     if (suppressedFields.length > 0) {
5         throw new RuntimeException("Attempting to bind disallowed fields: "
6             + StringUtils.arrayToCommaDelimitedString(suppressedFields));
7     }
8     medicamentService.addMedicament(newMedicamentToAdd);
9     return "redirect:/medicaments/all";
10 }

```

El codi del projecte "stmedioc" en l'estat **llista blanca al formulari** es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament és millor partir del que ja heu fet servir i copiar-lo amb el nom de "stmedioc402".

Fixeu-vos que hem afegit el paràmetre `BindingResult result`. Aquest paràmetre ens serveix per determinar els camps continguts a la petició i són a la *Whitelist* (`result.getSuppressedFields`). Recordeu que no voleu que hi siguin, i per això llanceu una excepció.

Si executeu l'aplicació per mostrar el formulari, l'ompliu i l'envieu, haureu d'obtenir l'excepció (vegeu la figura 4.4).

FIGURA 4.4. Sortida per a l'excepció de la 'Whitelist'



Traieu aquest camp del formulari de la vista `addMedicament.jsp` i veureu que treballa normalment sense donar l'excepció.

4.3 Estoc de medicaments, pàgina de 'login'

L'aplicació Estoc de medicaments mostra la llista de medicaments i permet crear-ne, però normalment no és convenient que qualsevol usuari connectat pugui afegir-hi medicaments.

Fareu que l'aplicació Estoc de medicaments tingui una pàgina de *login* que es cridi quan l'usuari intenti anar a la pàgina d'afegir medicaments. Aquesta pàgina de *login* autenticarà l'usuari mitjançant un codi d'usuari i una contrasenya, i segons qui sigui l'autoritzarà a anar a la pàgina d'afegir medicaments (**control d'accés**).

Per a aquesta implementació fareu servir funcionalitat Spring Security, afegint l'autenticació bàsica en les nostres pàgines web.

4.3.1 Configurant Spring Security

Spring Security és un *framework* que permet configurar i personalitzar l'autenticació i el control d'accés en aplicacions Spring.

Per fer servir Spring Security a la vostra aplicació heu de configurar les dependències del *framework* que afegiran les llibreries necessàries.

Afegiu les dependències que es mostren en el fitxer `pom.xml`.

A Estoc de medicaments es fa servir part de la funcionalitat de Spring Security, tot un *framework* que proporciona autenticació i control d'accés configurable: bit.ly/2nywYZy.

El codi del projecte "stmedioc" en l'estat **pàgina de login** es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament és millor partir del que ja heu fet servir i copiar-lo amb el nom de "stmedioc403".

```
1 <dependency>
2   <groupId>org.springframework.security</groupId>
```

```

3      <artifactId>spring-security-config</artifactId>
4      <version>3.1.4.RELEASE</version>
5      <exclusions>
6          <exclusion>
7              <artifactId>spring-asm</artifactId>
8              <groupId>org.springframework</groupId>
9          </exclusion>
10     </exclusions>
11 </dependency>
12 <dependency>
13     <groupId>org.springframework.security</groupId>
14     <artifactId>spring-security-web</artifactId>
15     <version>3.1.4.RELEASE</version>
16 </dependency>

```

Recordeu que també les podeu afegir amb l'assistent de NetBeans a tal efecte, la qual cosa us pot ajudar a trobar la versió més recent.

A la carpeta *WEB-INF/spring*, creeu l'arxiu *security-context.xml* amb el codi que es mostra.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xmlns:mvc="http://www.springframework.org/schema/mvc"
6       xmlns:security="http://www.springframework.org/schema/security"
7       xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www
8       .springframework.org/schema/mvc/spring-mvc.xsd
9       http://www.springframework.org/schema/security http://www.
10      springframework.org/schema/security/spring-security.xsd
11      http://www.springframework.org/schema/beans http://www.
12      springframework.org/schema/beans/spring-beans.xsd
13      http://www.springframework.org/schema/context http://www.
14      springframework.org/schema/context/spring-context.xsd">
15     <security:http auto-config="true" use-expressions="true">
16         <security:intercept-url pattern="/medicaments/add" access="hasRole('
17             adminRol')"/>
18         <security:form-login login-page="/login" default-target-url="/
19             medicaments/add" login-processing-url="/j_spring_security_check"
20             username-parameter="j_username" password-parameter
21             ="j_password" authentication-failure-url="/
22             loginfailed"/>
23         <security:logout logout-success-url="/logout" />
24     </security:http>
25     <security:authentication-manager>
26         <security:authentication-provider>
27             <security:user-service>
28                 <security:user name="ioc" password="ioc123" authorities="
29                     adminRol" />
30             </security:user-service>
31         </security:authentication-provider>
32     </security:authentication-manager>
33 </beans>

```

L'etiqueta *security:http*, els seus atributs i el seu contingut defineixen el veritable comportament de la seguretat.

L'etiqueta i els atributs

```

1 <security:intercept-url pattern="/medicaments/add" access="hasRole('adminRol')
  "/>

```

indiquen que una petició sobre */medicaments/add* serà interceptada i només hi podrà accedir qui hagi estat identificat amb *rol adminRol*.

L'etiqueta i els atributs

```
1 <security:form-login ...>
```

defineixen les pàgines que controlen l'autenticació efectiva, la fallida i el nom dels camps de *login*.

L'etiqueta i l'atribut

```
1 <security:logout logout-success-url="/logout" />
```

defineixen la pàgina de desconnexió.

L'etiqueta i els atributs

```
1 <security:user name="ioc" password="ioc123" authorities="adminRol" />
```

defineixen els usuaris i les seves credencials. L'atribut *authorities* determina el rol de l'usuari. En aquest cas només heu configurat un usuari, però se'n poden configurar més repetint l'etiqueta.

Completeu la configuració a nivell de l'aplicació. A *web.xml*, afegiu la següent configuració dins de *<web-app>*:

```
1 <context-param>
2   <param-name>contextConfigLocation</param-name>
3   <param-value>/WEB-INF/spring/security-context.xml</param-value>
4 </context-param>
5 <listener>
6   <listener-class>org.springframework.web.context.ContextLoaderListener</
   listener-class>
7 </listener>
8 <filter>
9   <filter-name>springSecurityFilterChain</filter-name>
10  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</
   filter-class>
11 </filter>
12 <filter-mapping>
13   <filter-name>springSecurityFilterChain</filter-name>
14   <url-pattern>/*</url-pattern>
15 </filter-mapping>
```

Amb el paràmetre de context *<context-param>* esteu indicant el nom del fitxer on heu configurat la seguretat. Com que el *listener* és *ContextLoaderListener*, es carregarà la configuració indicada a *security-context.xml* a l'inici de l'aplicació i en aquest nivell, sense esperar cap petició i sense dependre del *Dispatcher Servlet*.

Sense entrar en més detalls de Spring Security, el filtre està indicant que qualsevol petició (*<url-pattern>*) serà manegada pel filtre *springSecurityFilterChain*, i fent servir *DelegatingFilterProxy* esteu deixant a Spring que crei els *beans* necessaris.

4.3.2 El controlador per al 'login'

A la configuració que heu fet a `security-context.xml` heu assignat les URL que corresponen al *login*, al *logout* i a les credencials errònies. Són, respectivament, */login*, */logout* i */loginfailed*.

Amb aquesta informació, al paquet `cat.xtec.ioc.controller` podeu crear un nou controlador `LoginController` amb el codi que es mostra a continuació:

```
1 package cat.xtec.ioc.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestMethod;
7
8 @Controller
9 public class LoginController {
10
11     @RequestMapping(value = "/login", method = RequestMethod.GET)
12     public String login() {
13         return "login";
14     }
15
16     @RequestMapping(value = "/loginfailed", method = RequestMethod.GET)
17     public String loginerror(Model model) {
18         model.addAttribute("error", "true");
19         return "login";
20     }
21
22     @RequestMapping(value = "/logout", method = RequestMethod.GET)
23     public String logout(Model model) {
24         return "login";
25     }
26 }
```

Quan algú intenti accedir a l'URL */medicaments/add*, Spring Security mostrarà la pàgina *login.jsp*. Si l'usuari no s'autentica o no està en el rol adient, segons la configuració actuarà */loginfailed*, i s'hi afegirà l'atribut `error`.

Si l'usuari provoca una desconexió, llavors actuarà */logout* i es mostrarà un altre cop la pàgina de *login*.

4.3.3 La vistes

Com que un cop connectats l'aplicació us dirigirà a la vista *addMedicament*, podeu afegir aquí la desconexió.

Afegiu el codi següent a *addMedicament.jsp* dins de la *div* de classe *jumbotron*.

```
1 <a href="<c:url value="/j_spring_security_logout" />" class="btn btn-danger btn-
  mini pull-right">desconnectar</a>
```

A *views*, creeu la vista *login.jsp* amb el codi que es mostra a continuació:

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
3 <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
4 <%@page contentType="text/html" pageEncoding="UTF-8"%>
5 <!DOCTYPE html>
6 <html>
7     <head>
8         <meta http-equiv="Content-Type" content="text/html; charset=ISO
9             -8859-1">
10        <link
11            rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css
12            /bootstrap.min.css">
13        <title>Medicaments</title>
14    </head>
15    <body>
16        <section>
17            <div class="jumbotron">
18                <div class="container">
19                    <h1>Medicament</h1>
20                    <p>Afegir medicament</p>
21                </div>
22            </div>
23        </section>
24        <div class="container">
25            <div class="row">
26                <div class="col-md-4 col-md-offset-4">
27                    <div class="panel panel-default">
28                        <div class="panel-heading">
29                            <h3 class="panel-title">Si us plau, proporcioni les
30                                seves dades</h3>
31                        </div>
32                        <div class="panel-body">
33                            <c:if test="${not empty error}">
34                                <div class="alert alert-danger">
35                                    Credencials incorrectes
36                                </div>
37                            </c:if>
38                            <form action="<c:url value= "/"
39                                j_spring_security_check"> </c:url>" method="
40                                post">
41                                <fieldset>
42                                    <div class="form-group">
43                                        <input class="form-control" placeholder
44                                            ="Usuari" name='j_username' type="
45                                            text">
46                                    </div>
47                                    <div class="form-group">
48                                        <input class="form-control" placeholder
49                                            ="Contrasenya" name='j_password'
50                                            type="password">
51                                    </div>
52                                    <input class="btn btn-lg btn-success btn-
53                                        block" type="submit" value="Connectar
54                                        ">
55                                </fieldset>
56                            </form>
57                        </div>
58                    </div>
59                </div>
60            </div>
61        </div>
62    </body>

```

L'estructura de selecció `<c:if ...>` mostrarà el missatge en el cas que l'atribut `error` no sigui buit. Recordeu que el controlador l'omplirà en cas que l'usuari no s'hagi autenticat.

L'acció del formulari es correspon a la configuració feta a `security-context.xml`, concretament l'atribut `login-processing-url="/j_spring_security_check"`, i els noms dels camps "usuari" i "contrasenya" també es corresponen amb la configuració.

4.3.4 Provant d'entrar

Executeu l'aplicació amb l'URL `localhost:8080/stmedioc403/medicaments/add`.

Veureu la pàgina de *login* tal com es mostra en la figura 4.5.

FIGURA 4.5. Formulari de 'login' (autenticació)



Medicament
Afegeix medicament

Si us plau, proporcioneu les vostres dades

Usuari

Contrasenya

Connecta

Proveu amb dades no correctes i l'aplicació us mostrarà el missatge de credencials no vàlides (vegeu la figura 4.6).

FIGURA 4.6. Formulari de 'login' (autenticació), credencials incorrectes



Medicament
Afegeix medicament

Si us plau, proporcioneu les vostres dades

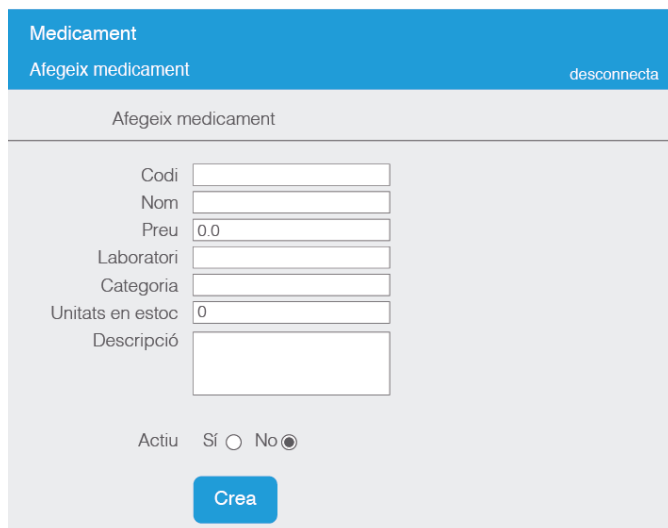
Credencials incorrectes

Usuari

Contrasenya

Connecta

Proveu, doncs, amb credencials correctes, i llavors us mostrarà la pàgina per afegir el medicament. Fixeu-vos que hi ha l'enllaç a desconnexió (vegeu la figura 4.7).

FIGURA 4.7. Formulari de medicament amb enllaç a desconnexió

Medicament

Afegeix medicament desconnecta

Afegeix medicament

Codi

Nom

Preu

Laboratori

Categoria

Unitats en estoc

Descripció

Actiu Sí ☐ No ☒

Amb aquest exemple heu assegurat una pàgina de la vostra aplicació, i es pot interpretar la generalització a qualsevol pàgina de l'aplicació.

Heu fet servir **Spring Security** baixant les dependències i l'heu configurat amb l'arxiu `security-context.xml`, on heu definit el comportament de Spring Security quant a:

- Quines pàgines s'ha d'assegurar (interceptar) i quins rols estaran autoritzats a accedir-hi.
- Quines són les pàgines i URL de *login*, *logout* i *error*.
- Quin és el formulari que feu servir per a l'autenticació.
- Quins són els camps d'usuari i contrasenya que conté el formulari.
- Quins usuaris hi ha a l'aplicació.

La configuració d'aquest arxiu i el *bean* que manega la seguretat es carreguen a l'inici de l'aplicació, i serà vàlida per a tot el context de l'aplicació segons el que s'ha indicat a `web.xml`.

Heu implementat un controlador que manega les peticions segons les URL de l'arxiu de configuració.

Finalment, heu implementat la vista amb el formulari d'autenticació (*login*).

4.4 Estoc de medicaments, internacionalització

L'aplicació Estoc de medicaments està practicament completada segons els objectius pedagògics que s'havien proposat. Però avui dia les aplicacions han de ser

multiidioma, i per això cal dotar la nostra aplicació de la possibilitat de mostrar-se en diversos idiomes.

No tan sols això, sinó que es parlarà d'una **internacionalització** que permet definir més paràmetres, com quina moneda es fa servir i quin és el separador decimal, entre altres personalitzacions.

Fareu servir la funcionalitat que es coneix com a **externalització de missatges** a Spring MVC per mostrar el text de les etiquetes en diversos idiomes, segons arxius de propietats.

L'externalització és el pas previ a la internacionalització, ja que cal tenir el text de les etiquetes dels diferents idiomes en diferents arxius de configuració per tal que la funcionalitat d'*i18N* vagi a buscar-los.

Un cop externalitzats els missatges, ja es pot configurar i implementar tot el que és relatiu a *i18n*. Per detectar i canviar d'idioma fareu servir `SessionLocaleResolver` a nivell de *web application context* i l'interceptor `LocaleChangeInterceptor` també en aquest nivell.

La **internacionalització** s'abreia sovint com "i18n" (de l'anglès *I-eighteen letters-N*), que prové de la "i" del començament, les 18 lletres següents i la "n" del final.

El codi del projecte "stmedioc" en l'estat **internacionalització** es pot descarregar des de l'enllaç que trobareu als annexos de la unitat. Però per seguir el desenvolupament és millor partir del que ja heu fet servir i copiar-lo amb el nom de "stmedioc404".

4.4.1 Externalitzem els missatges

Per externalitzar els missatges de la vostra aplicació i després poder traduir-los i fer-los servir en diversos idiomes fareu canvis en diversos textos d'algunes etiquetes del formulari de la vista `addMedicament.jsp`.

Per altra banda, creareu diversos fitxers de propietats, un per defecte i altres per a altres idiomes (concretament, un addicional per a l'anglès).

A més, haureu de configurar un *bean* de Spring per connectar la vista amb els fitxers de propietats que contenen els textos.

A `addMedicament.jsp`, modifiqueu les etiquetes de *code*, *name*, *price*, *producer* i *category* substituint el text, respectivament, per les etiquetes Spring que s'hi mostren.

```
1 <spring:message code= "addMedicament.form.medicamentId.label"/>
2 <spring:message code= "addMedicament.form.name.label"/>
3 <spring:message code= "addMedicament.form.price.label"/>
4 <spring:message code= "addMedicament.form.producer.label"/>
5 <spring:message code= "addMedicament.form.category.label"/>
```

Amb l'etiqueta `spring:message`, mitjançant l'atribut *code*, esteu associant el text amb la propietat definida al fitxer de propietats que creareu per a cada idioma de l'aplicació.

messages.properties

A la carpeta `/src/main/resources` (el que veieu dins d'*Other Sources* en la pestanya de projectes de NetBeans), creeu un arxiu de propietats de nom `messages.properties`. Serà l'arxiu per defecte, és a dir, que quan no trobi cap idioma serà el que farà servir.

Afegiu-hi les propietats de cada etiqueta.

```
1 addMedicament.form.medicamentId.label = Codi de medicament
2 addMedicament.form.name.label = Nom
3 addMedicament.form.price.label = Preu
4 addMedicament.form.producer.label = Laboratori – Fabricant
5 addMedicament.form.category.label = Categoria
```

Finalment, heu de configurar el *bean* que connecta el fitxer de propietats amb les etiquetes `spring:message` que heu afegit al jsp. Es configura a nivell de *web application context*, és a dir, afegiu un *bean* a l'arxiu `DispatcherServlet-servlet.xml`.

```
1 <bean id= "messageSource" class="org.springframework.context.support.
   ResourceBundleMessageSource">
2 <property name="basename" value="messages"/>
3 </bean>
```

L'atribut `value="messages"` és el que determina que el fitxer de propietats és *messages*.

Executeu l'aplicació amb l'URL `localhost:8080/stmedioc404/medicaments/add` per mostrar el formulari. Això sí, us haureu d'identificar i, un cop fet, us mostrarà el formulari amb els valors de l'arxiu de propietats (vegeu la figura 4.8).

FIGURA 4.8. Formulari amb textos externalitzats

The screenshot shows a web application interface for adding a medication. The header is blue with the text 'Medicament' and 'Afegeix medicament'. Below this is a light gray bar with the text 'Afegeix medicament'. The main form area is white and contains the following fields:

- Codi:
- Nom:
- Preu:
- Laboratori - Fabricant:
- Categoria:
- Unitats en estoc:
- Descripció:

At the bottom of the form, there are two radio buttons: 'Actiu' with 'Sí' selected, and 'No' with 'No' selected. Below these is a blue button labeled 'Crea'.

4.4.2 Configurant i implementant el canvi d'idioma

A *resources*, creeu l'arxiu *messages_en.properties* per poder tenir dos idiomes. Afegiu-hi les propietats de *messages.properties*, però canvieu els valors.

```
1 addMedicament.form.medicamentId.label = Medicament ID
2 addMedicament.form.name.label = Name
3 addMedicament.form.price.label = Price
4 addMedicament.form.producer.label = Producer
5 addMedicament.form.category.label = Category
```

A la vista *addMedicament.jsp*, afegiu el **selector** per canviar d'idioma just després de l'enllaç de desconnexió.

```
1 <div class="pull-right" style="padding-right:50px">
2 <a href="?language=ca" >Català</a>|<a href="?language=en" >Anglès</a>
3 </div>
```

A l'arxiu de configuració de *web application context* *DispatcherServlet-servlet.xml*, afegiu la definició del *bean*.

```
1 <bean id="localeResolver" class="org.springframework.web.servlet.i18n.
  SessionLocaleResolver">
2 <property name="defaultLocale" value="ca"/>
3 </bean>
```

SessionLocaleResolver és l'objecte que assigna atributs locals sobre la sessió de l'usuari. Una de les propietats que conté l'objecte és *defaultLocale*. En el vostre cas, dieu que el llenguatge per defecte és el català (*ca*).

També a *DispatcherServlet-servlet.xml*, heu d'afegir l'interceptor *LocaleChangeInterceptor* tal com es mostra a continuació:

```
1 <mvc:interceptors>
2   <bean class="org.springframework.web.servlet.i18n.
      LocaleChangeInterceptor">
3     <property name="paramName" value="language"/>
4   </bean>
5 </mvc:interceptors>
```

Amb aquest interceptor esteu dient que el nom del paràmetre que correspon al canvi d'idioma és *language*, i això es correspon amb el que heu afegit a *addMedicament.jsp*.

4.4.3 Provant a canviar d'idioma

Executeu l'aplicació amb l'URL *localhost:8080/stmedioc404/medicaments/add* i, un cop autenticats satisfactòriament, us mostrarà el formulari (vegeu la figura 4.9).

FIGURA 4.9. Formulari amb els textos en l'idioma per defecte

The screenshot shows a web form titled 'Medicament' with a blue header. Below the header, there is a navigation bar with 'Afegeix medicament', 'Català | Anglès', and 'desconnecta'. The main form area has a title 'Afegeix medicament' and several input fields: 'Codi', 'Nom', 'Preu' (with '0.0' entered), 'Laboratori', 'Categoria', 'Unitats en estoc' (with '0' entered), and 'Descripció'. At the bottom, there is a radio button group for 'Actiu' with 'Sí' and 'No' (selected) options, and a blue 'Crea' button.

Fixeu-vos que les etiquetes són en català. Si premeu l'enllaç *Anglès*, llavors les etiquetes canviaran (vegeu la figura 4.10).

FIGURA 4.10. Formulari amb els textos en anglès

The screenshot shows the same web form as Figure 4.9, but with the text in English. The header is 'Medicament', the navigation bar has 'Afegeix medicament', 'Català | Anglès', and 'desconnecta'. The main form area has a title 'Afegeix medicament' and input fields: 'MedicamentID', 'Name', 'Price' (with '0.0' entered), 'Producer', 'Category', 'Unitats en estoc' (with '0' entered), and 'Descripció'. At the bottom, there is a radio button group for 'Actiu' with 'Sí' and 'No' (selected) options, and a blue 'Crea' button.

Si torneu a prémer *Català* tornarà a mostrar les etiquetes en aquest idioma. També podeu veure que les URL contenen el paràmetre `language`.

4.5 Què s'ha après?

Per veure com es genera i manega a Spring MVC un flux de dades des de la vista fins al model heu afegit el formulari per crear medicaments.

Al controlador, al mètode que desencadena mostrar el formulari, es crea un atribut que passarà a la resposta amb un *domain object* (*Medicament*) buit. A la vista, a l'etiqueta `form:form`, mitjançant l'atribut d'etiqueta `modelAttribute`,

es relacionen els camps del formulari amb les propietats de l'objecte. Canvis en els valors del formulari provoquen canvis en l'objecte, i per això s'anomena *form-backing bean*.

El mètode que recull el POST del formulari amb l'anotació:

```
1 @ModelAttribute("newMedicament") Medicament newMedicamentToAdd
```

està recollint l'objecte *form-backing bean* i el podrà fer servir amb la variable declarada (*newMedicamentToAdd*).

Heu afegit una llista de propietats (*Whitelist*) per obviar un dels camps del model (*stockInOrder*). Ho heu aconseguit amb l'anotació *@InitBinder* i la implementació del mètode que la segueix al controlador específic.

Heu fet servir **Spring Security**, un *framework* que permet configurar i personalitzar l'autenticació i el control d'accés en aplicacions Spring. Per usar el *framework* heu desenvolupat la pàgina de *login*, afegint l'arxiu de configuració de la seguretat *security-context.xml*, on heu definit els usuaris i el comportament de la vostra aplicació respecte a la seguretat (per exemple, quines pàgines necessitaran autenticació i quines pàgines han de mostrar-se per a *login*, *logout* i *error*).

Finalment, heu fet que la vostra aplicació tingui **internacionalització** (*i18n*). Heu externalitzat missatges ajudats pels fitxers de propietats i etiquetes específiques als *jsp* (*spring:message*) i heu configurat la internacionalització amb els *beans* necessaris a nivell de *web application context*: *SessionLocaleResolver* i *LocaleChangeInterceptor*.