

SHAKE THE FUTURE



Web Programming

SPRING and JPA

JY Martin - JM Normand

Plan

- 1 Objectives
- 2 SPRING
- 3 Create SPRING project
- 4 Developping our web application

Main objective

In this practical work, we will work on JAVA SPRING Framework.
We will also use JPA as ORM.

The main objective is to use a servlet based framework.

Why a JAVA Servlet Framework?

- JAVA is a structured language with many usefull concepts
- Can be use on most environments with a single development
- Compiled. That avoid a lot problems.
- Can handle big projects.

We use SPRING because it is one of the most used in a JAVA environment.

Tools

To write this application you will need an IDE : Netbeans. You can also use Eclipse or IntelliJ but these slides are designed for Netbeans.

To display files, you will need a web browser with debugging tools.

You will need a Servlet server. We will use Tomcat. We use Tomcat 8.5, but it should be ok with newer versions.

And of course a Database Server. We will use PostgreSQL.

Have a look to prerequisites to install and to check tools.

Plan

- 1 Objectives
- 2 SPRING**
- 3 Create SPRING project
- 4 Developping our web application

Servlet

A **Servlet** is a JAVA based service that replies to HTTP requests.

It uses:

- a `HttpServletRequest` to get the request -the user request-
- a `HttpServletResponse` to reply to the request -the server response-

Basic servlet based applications are called J2EE or J2E.

JSP

Java **S**erver **P**ages is a HTML based syntax that can include JAVA code to build replies.

These pages are used by servlet servers to serve HTML pages.

Servlet based frameworks

Servlet are not really very friendly to use.

Some framework were introduced, based on servlets, to facilitate application development.

STRUTS, SPRING, SPRINGBoot are such frameworks.

About SPRING

SPRING and SPRINGBoot are 2 JAVA Servlet based frameworks. They are based on the same concepts. SPRINGBoot was created over SPRING to facilitate configurations. It also includes its own servlet server.

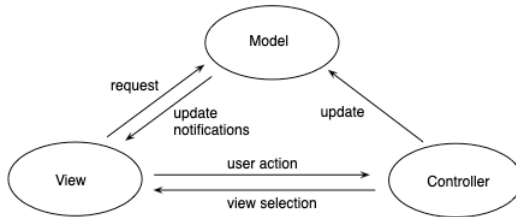
You can have a look to these links:

- <https://spring.io>
- <https://spring.io/projects/spring-boot>

We will use SPRING.

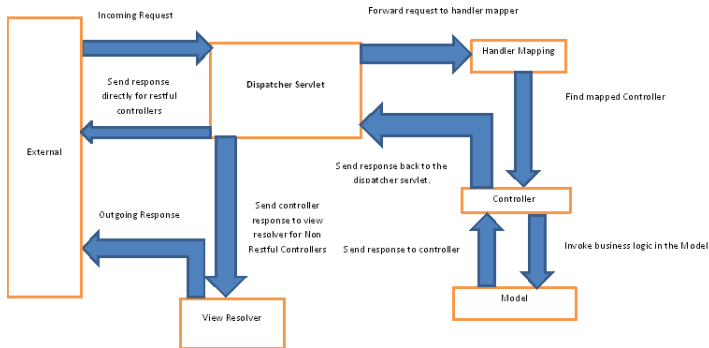
SPRING MVC

SPRING implements the MVC architectural pattern.



- Model is Data modeling
- View is the way to present informations
- Controller computes and defines view. Also, it interacts with Data modeler.

How does it works 1/2



From <https://www.baeldung.com>

How does it works 2/2

- Dispatcher receives request
- Dispatcher asks to the Handler Mapping to call the right controller.
- Handler Mapping search for the controller sends request to it
- Controller processes request, defines de the view model to implement
- Controller sends view model and data to the Dispatcher
- Dispatcher sends the view model and data to the View Resolver.
- View Resolver processes view and data to produce a response.
- View Resolver sends response to user.

annotations

We will manage SPRING through annotations.

Annotations are instructions that are processed and stored by the JAVA compiler and that can be used by the application.

They implement interface Annotation from java.lang.annotation

In source code, they are defined with char @ followed by the annotation, and may have parameters.

Here are some examples:

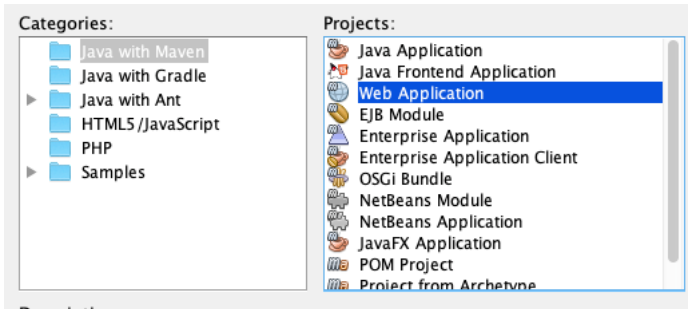
- @Override
- @Controller
- @RequestMapping("index.do")
- @RequestMapping(value="index.do",method=RequestMethod.POST)

Plan

- 1 Objectives
- 2 SPRING
- 3 Create SPRING project
- 4 Developping our web application

Create a SPRING project in Netbeans 1/7

Open Netbeans and create a new project. Select the “Java With Maven” project and “Web Application”.



Create a SPRING project in Netbeans 2/7

Give name and select project location.

Name and Location

Project Name:

Project Location:

Project Folder:

Artifact Id:

Group Id:

Version:

Package: (Optional)

- our group ID will be "org.centrale"
- our version is "1.0-SNAPSHOT"
- we create a package "org.centrale.prweb"

Go next.

Create a SPRING project in Netbeans 3/7

Select server and JAVA EE version.



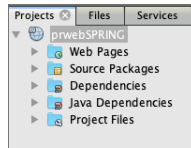
We choose Tomcat with JAVA EE 7 Web.

If Tomcat is not declared in the list, select "add" and add Tomcat Path to Netbeans.

Validate.

Create a SPRING project in Netbeans 4/7

Here is what you should have in your projet tab.

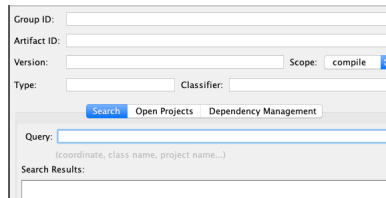


Create a SPRING project in Netbeans 5/7

Now let's add dependencies. Right clic on "Dependencies" to add new dependencies.



Here is the window.



Create a SPRING project in Netbeans 6/7

In the Query field enter "SPRING web mvc".

You should have many results.

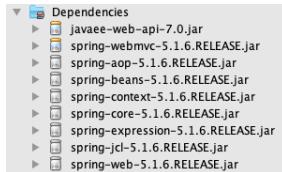
Select "org.springframework : spring-webmvc".

Open it. and select recent jar version. 5.1.6 would be ok.

Validate.

Create a SPRING project in Netbeans 7/7

Now open your “Dependencies” folder. Many dependencies were added. Maven checks for dependencies and include them.



Right clic on “Dependencies”. “Download Declared Dependencies” to be sure they are loaded.

POM file

In your project, have a look to "Project Files/pom.xml".

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.6.RELEASE</version>
  </dependency>
  <dependency>
```

This file is a MAVEN file. It stores every dependency you required for the project.

This is a text/xml file, so if you make any change to this file, you will have to reload dependencies like you did in previous slide.

Check SPRING project

Build your project and run it.

The first time you run it since you start Netbeans, you should be asked for your admin login/password in Tomcat.

Your browser should be launched with this page as a result:



Hello World!

We **STRONGLY** recommend cleaning and building your project regularly.

Project compilation and redeployment are not always guarantee when you modify something.

What happens ?

- Netbeans build your project and create a WAR file. Have a look to the target directory in your project.
Web ARchive files are compressed file that contains your compiled project, with classes, html, configs, ...
- Netbeans add WAR file to Tomcat applications. Tomcat decompress WAR file and launch it
- Netbeans start your browser with the project URL.

then:

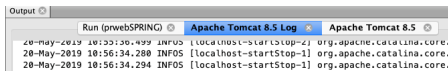
- Your browser send URL to tomcat
- your project send back the index.html file.

This index.html file is in the one in "Web Pages" in your project

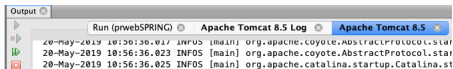
Output interface

Bottom of the Netbeans window, you should have an output tag. In this tag, several tags.

- The log tag: that includes tomcat logs. Errors are logged here, so do not forget to have a look for errors inside this tag.



- The server tag: it displays information from the server. See the small red icon on the left? it is used to stop the server.



What about SPRING ?

OK, but what about SPRING, the Dispatcher, and so on ?

We have to do a bit of configuration for that.

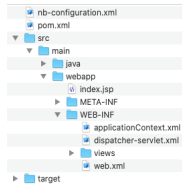
Configure project for SPRING 1/7

Decompress SPRING.zip from the materials.

- From your System interface (we recommend this because creating it through netbeans interface is not guarantee)
 - In your src/main/webapp folder, create a new **WEB-INF** folder.
 - In your WEB-INF folder, create a **views** folder. This folder will contains our views.
 - Copy the 3 xml files from materials to WEB-INF:
 - applicationContext.xml
 - dispatcher-servlet.xml
 - web.xml
 - Rename index.html to index.jsp

Configure project for SPRING 2/7

here is the structure you may have for your project.



Configure project for SPRING 3/7

Have a look to the way these 3 xml files are build.

- like HTML files, tags are imbricated. This is the structure all XML files.
- first tag is the root. It contains many attributes. For example:
 - xmlns:mvc means we can use prefix "mvc" for tags; It also gives the "XML schema" for it.
 - xsi:schemaLocation tells where we can find which schema -XML strutures definitions- through pairs of values.

It your tag is prefixed, you should find its declaration in the root tag.

Configure project for SPRING 4/7

What do you find in these XML files?

- **web.xml** : global configuration file for servlet applications.
 - context-param: Application configuration is in /WEB-INF/applicationContext.xml
 - listener: Should listen for spring events
 - servlet: we define a servlet : the dispatcher.
 - servlet-mapping: Dispatcher manages all ".do" requests
 - welcome-file-list: startup file is index.jsp

Configure project for SPRING 5/7

- **dispatcher-servlet.xml**: Dispatcher configuration
 - bean id="viewResolver": views are prefixed by /WEB-INF/views/ and suffixed by .jsp
View called **index** is file /WEB-INF/views/**index.jsp**
- **applicationContext.xml**: Application configuration
 - mvc:annotation-driven: We manage application through annotations
 - context:component-scan: Controllers are in "org.centrale.prweb.controllers"

We will add some more informations in the applicationContext.xml file, a bit later.

Configure project for SPRING 6/7

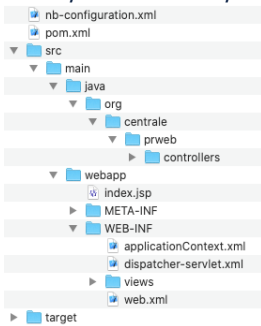
We said in `applicationContext.xml` that our controllers are in `"org.centrale.prweb.controllers"`.

So, in your Netbeans Project, add a package to your sources:
`org.centrale.prweb.controllers`

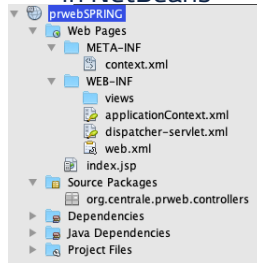
Configure project for SPRING 7/7

Here is the structure you may have.

In your directory



In NetBeans



Run it

OK. Let's see it still works.

If your server is still running, stop it. Run project. You should still have this:



Hello World!

Out first controller 1/3

From materials,

- copy file **index.jsp** to the views
- Copy file **StartupController.java** to the controllers folder
- Change the index.jsp file located **root of the web pages** by this:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>  
<% response.sendRedirect("index.do"); %>
```

This redirects the index.jsp call to index.do so that the dispatcher will catch it.

Do not copy/paste these lines. PDF files contains hidden characters that will lead to errors in your file.

Out first controller 2/3

StartupController.java is our first controller.

```
@Controller
public class StartupController {
    @RequestMapping("index.do")
    public ModelAndView home() {
        return new ModelAndView("index");
    }
}
```

- Annotation **Controller** means this class is a controller
- Annotation **RequestMapping** tells which method handles the request and the route to handle: index.do
- **ModelAndView** is the class that defines the view model. Be sure you import **servlet** ModelAndView and not **portlet** one.
- the ModelAndView that is returned indicates, in our view folder, which one have to be used.

Out first controller 3/3

Ok, run project once more.
It should have change to this:

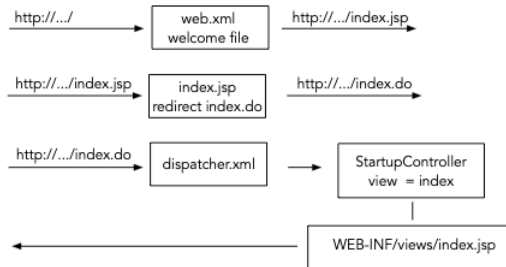


If you have an error:

- Maybe you should consider "Clean And Build" before running.
- Maybe you should have a look to applicationContext.xml.
There should be something wrong in your controllers declaration.
- If you do not have the right page, check you have 2 index.jsp files with the right contents.

What have we done ?

Here is the way it works.



What have we done ?

- We told Tomcat
 - applicationContext.xml is our application configuration file
 - we use annotation to manage everything
 - controllers are in org.centrale.prweb.controllers
 - There is a dispatcher
 - views are in WEB-INF/views and ends with .jsp
- We redirect index.jsp to index.do so that dispatcher catches it and manages it
- We build a controller that manages index.do routes. This controller calls the index view.
- Dispatcher converts index view to file
WEB-INF/views/index.jsp
- At last it sends data

How annotations works

In our controller we use annotations.

- @controller tells the compiler that the class is a controller and that it may handle routes.
- @RequestMapping tells the compiler that the following method handles a route, index.do.

It creates invisible code lines for the dispatcher: “if URL is index.do, for GET and POST methods, call this method”.

Annotations are used to implement AOP: **Aspect Oriented Programming**.

We will meet annotations several times during this practical work.

What happened when we launched the project.

1/2

- As the previous time, project is defined in tomcat and decompressed. Browser is called with the default route for the project.
- Tomcat knows default route is index.jsp, so it gets it.
- index.jsp redirects to index.do
- Application knows *.do routes are managed by the dispatcher. It sends the request to the dispatcher

What happened when we launched the project.

2/2

- dispatcher knows, thanks to the annotation, that index.do is managed by "home" in "StartupController", so it calls the method.
- StartupController defines index as the resulting view and creates the appropriate object.
- dispatcher send the requested view to the View Resolver.
- View resolver knows views are prefixed by /WEB-INF/views and suffixed by jsp. It gets the file and processed it to create the html view.
- View resolver sends html back to user

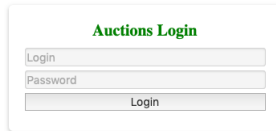
Plan

- 1 Objectives
- 2 SPRING
- 3 Create SPRING project
- 4 Developping our web application

Creating a controller 1/7

Let's start with a basic controller: an Identification controller.

We will use this page:



A screenshot of a web form titled "Auctions Login" in green text. The form contains two input fields: "Login" and "Password", both with light gray borders. Below these fields is a "Login" button with a light gray background and a thin border.

You should already have developed this page in the HTML practical work. If not, you will find one in the materials in authenticationpage.

We will use StartupController to build it.

Creating a controller 2/7

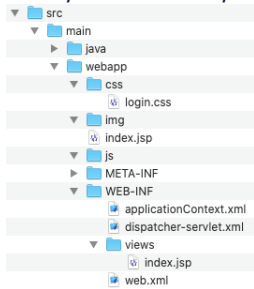
- If you use materials -in authenticationpage-:
 - Copy js, css and img folder from the materials to "the Web Pages" folder.
 - replace the index.jsp in views by the index.jsp in the materials.
- if you use your own files:
 - Remove index.jsp from the views.
 - Copy your html file to the views and rename it index.jsp
 - Link the form to the action index.do using POST.
 - Add this line as the first line of the file

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

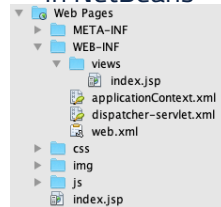
Creating a controller 3/7

JS files, CSS files, Image files, ... have to be placed in the "Web Pages" folder, the same level as "WEB-INF" and "META-INF". If they are located in folders, these folders have to be in "Web Pages".

In your directory

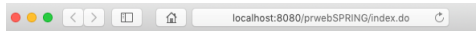


In NetBeans



Creating a controller 4/7

Run project and try it.

A screenshot of a web form titled "Auctions Login" in green text. Below the title are two input fields: "Login" and "Password". The "Login" field is highlighted with a blue border. Below these fields is a button labeled "Login".

When you give login/password, it restarts the url index.do
Why ?

Creating a controller 5/7

Our RequestMapping annotation handles the route index.do for GET **AND** POST methods.

We want to check login/password and if they are valid, display the information page. We need 2 methods: one for GET and one for POST.

Change the RequestMapping of your controller by this one:

```
@RequestMapping(value="index.do", method=RequestMethod.GET)
```

Creating a controller 6/7

This modification means the home method will handle GET mode only.

Maybe it would be nice to rename the method **handleGet**.

Try it once more.

Should work for GET and crash for POST. We also need a method for POST.

Creating a controller 7/7

Duplicate method `handleGet` and call it `handlePost`.

Change annotation before the `handlePost` method by:

```
@RequestMapping(value="index.do", method=RequestMethod.POST)
```

Try it once more.

It works.

- GET is handled by `handleGet`.
- POST is handled by `handlePost`.

Managing authentication

Ok, but how can we get informations from the form?

There are 2 ways of doing this:

- Using the `HttpServletRequest` object used by servlets and that contains the full request
- Using an aspect oriented programming method

Managing authentication - 1st method

Let's use the first method: `HttpServletRequest`.

Add a `HttpServletRequest` parameter to `handlePost`.

Get parameters from your request. Parameters name are the ones in your form in `index.jsp` -the view one-. Check you use the right parameters.

You might have something like this:

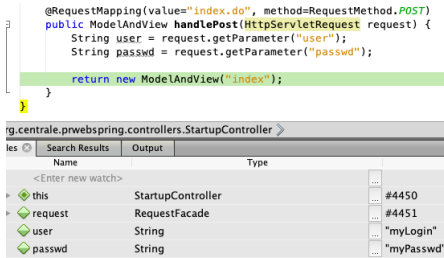
```
@RequestMapping(value="index.do", method=RequestMethod.POST)
public ModelAndView handlePost(HttpServletRequest request) {
    String user = request.getParameter("user");
    String passwd = request.getParameter("passwd");

    return new ModelAndView("index");
}
```

Managing authentication - 1st method

Place a breakpoint on the return line of the method `handlePost`.
Run project using **debug mode**.

Give a login and a password. Validate.
Project should stop at your breakpoint. Check variables `user` and `passwd`.



Managing authentication - 1st method - Explanation

We changed the `handlePost` method, adding a parameter, and it still works. Why ?

Remember AOP, the **Aspect Oriented Programming**?

`RequestMapping` add lines to the method following the annotation, but it does more: it looks for the method parameters and insert them in the invisible code. If compiler knows possible parameters, it uses them.

Managing authentication - 1st method - Explanation

HttpServletRequest, **HttpServletResponse** are standart kind of parameters, AOP can deal with them for the handlePost method. And it does.

getParameter is a HttpServletRequest method that retrieves a parameter from a request, that gets informations from a form. You give the name, it sends back the value, as a String. It's up to you to cast it to something else, if necessary.

Managing authentication - 2nd method

If dispatcher can give parameters, maybe it can do more.
We will use more advanced annotations from AOP.

Let's create a **User** class just beside our controller.

```
public class User {  
    private String user;  
    private String passwd;  
}
```

Add getters and setters for the 2 attributes.

Managing authentication - 2nd method

Come back to handlePost.

Replace the HttpServletRequest parameter by a **User** parameter.
Add annotation **@ModelAttribute("User")** just before User to tell we use User as a form model.

Keep your breakpoint.

```
@RequestMapping(value="index.do", method=RequestMethod.POST)
public ModelAndView handlePost(@ModelAttribute("User")User anUser) {
    String user = anUser.getUser();
    String passwd = anUser.getPasswd();

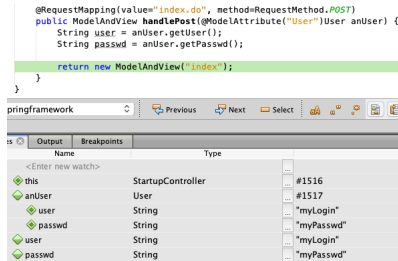
    return new ModelAndView("index");
}
```

Managing authentication - 2nd method

Let's go.

It compiles. Annotations link our User class and the parameters.
They generate code to get parameters from the request.
Run it in debug mode. Give login/password. Validate.

Program should pause. Have a look to the variables.



Managing authentication - 2nd method - Explanation

Using AOP also means: according to variable names, method names, deduce code to implement.

The ModelAttribute annotation preprocessed the source code and add invisible code:

- To create a User object. User is the information we gave to ModelAttribute.
- To give values to the attributes according to the available setters in User.

Managing authentication - 2nd method - Explanation



RequestMapping maps the route, look for parameters and... there is a User attribute to give to the method.

And you've got the result.

Managing authentication

If the user authenticates, we want him/her to get auctions.

List of items

Auction#	Title	Author	Body	
	Computer	JY Martin	I sell my old computer, 3 years old	
				

We will use the page you already developed to display auctions in the HTML exercise.

You can also find the page in the materials, in `auctionspage`.

Feel free to use the retrieving method you prefer to authenticate user.

Managing authentication

In method `handlePost`, we have to check if user login/password is valid -you can choose the values you want-.

Then:

- if it is valid, we return `ModelAndView "auctions"`
- if it is not, we go back to `"index"`

```
public ModelAndView handlePost(@ModelAttribute("User")User anUser) {  
    ModelAndView returned;  
    if ((anUser.getUser().equals("admin")) && (anUser.getPasswd().equals("admin"))) {  
        returned = new ModelAndView("auctions");  
    } else {  
        returned = new ModelAndView("index");  
    }  
    return returned;  
}
```

Managing authentication

Run project.

- check with bad login / password
- check with right login / password

Adding entities

Our auctions page should not be a static one. Data should come from a database.

Let's use an ORM and a persistence unit -JPA- to manage informations.

First, we need some more SPRING modules.
We will use spring-data-jpa.

Adding entities - adding modules

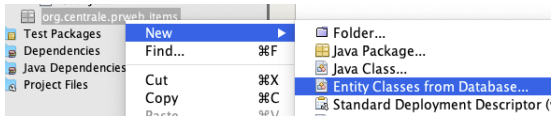
Add dependencies to your project. We add spring-data-jpa, a postgresql driver and tools for jsp files: jstl.

- search for spring-data-jpa
get it from org.springframework.data:spring-data-jpa
select version 2.1.7 jar
- search for postgresql
get it from org.postgresql:postgresql
select last jar version
- search for jstl
get it from jstl:jstl
select jar version 1.2 or 1.0.2

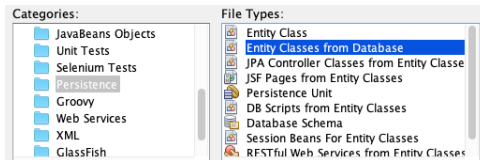
Adding entities

In your project, create a package “org.centrale.prweb.items”.

Right clic on the package. Select “New...” then “Entity classes from Database...”.

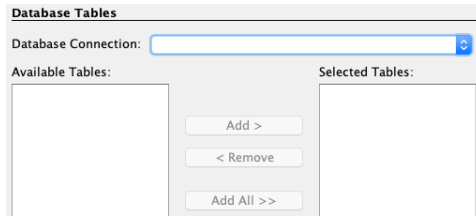


If you don't have it in your menu, select “Other...”



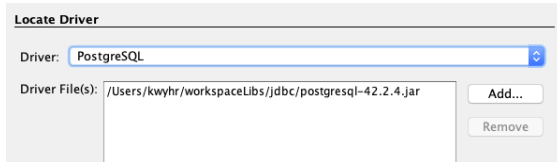
Adding entities

You should have this window:



Click on "Database Connection" and select "New Database Connection" to open a new window.

Adding entities -create connection-

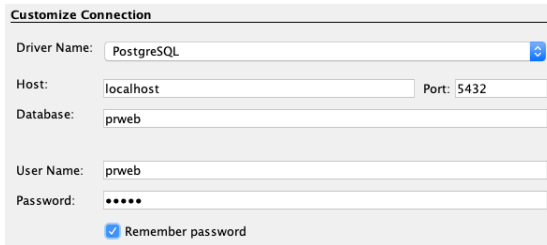


Select Driver “postgresql”. If there is none in the list, Click on “Add” to add one, and select a postgresql JDBC driver.

Click on “Next>”.

Adding entities -create connection-

Give your database information for the “prweb” database.



Customize Connection

Driver Name: PostgreSQL

Host: localhost Port: 5432

Database: prweb

User Name: prweb

Password: •••••

☒ Remember password

Maybe you should test the connection to avoid misspelling.

Click on “Next>”.

Adding entities -create connection-

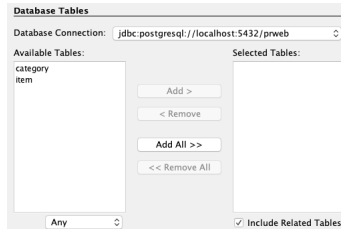
In the next screen, select the schema you will use. Surely "public".
Click on "Next>".

Give a name to your connection. The one you want, it's for you.

Give your database information for the "prweb" database. Click on
"Finish".

Adding entities

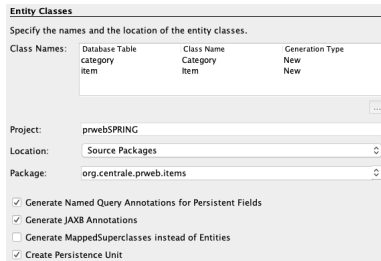
Netbeans connects to the database and retrieve tables.



"Add All" to add the 2 tables. They should now appear in the column "Selected Tables".
Click on "Next>".

Adding entities

Next screen is a summary of what Netbeans will do.



The 'Entity Classes' dialog box in NetBeans is shown. It has a title bar 'Entity Classes' and a subtitle 'Specify the names and the location of the entity classes.' Below this, there is a table for 'Class Names' with three columns: 'Database Table', 'Class Name', and 'Generation Type'. The table contains two rows: 'category' with 'Category' and 'New', and 'item' with 'Item' and 'New'. Below the table is a 'Project:' field with 'prwebSPRING' and a 'Location:' dropdown menu set to 'Source Packages'. Below that is a 'Package:' dropdown menu set to 'org.centrale.prweb.items'. At the bottom, there are four checkboxes: 'Generate Named Query Annotations for Persistent Fields' (checked), 'Generate JAXB Annotations' (checked), 'Generate MappedSuperclasses instead of Entities' (unchecked), and 'Create Persistence Unit' (checked).

Database Table	Class Name	Generation Type
category	Category	New
item	Item	New

Project: prwebSPRING

Location: Source Packages

Package: org.centrale.prweb.items

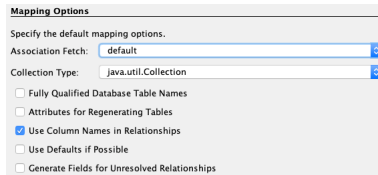
- ☒ Generate Named Query Annotations for Persistent Fields
- ☒ Generate JAXB Annotations
- ☐ Generate MappedSuperclasses instead of Entities
- ☒ Create Persistence Unit

You can still change some elements, like the package where Netbeans will generate classes.

If all is ok, click on "Next>".

Adding entities

Next screen defines the way mapping will act.



Mapping Options

Specify the default mapping options.

Association Fetch:

Collection Type:

☐ Fully Qualified Database Table Names

☐ Attributes for Regenerating Tables

☒ Use Column Names in Relationships

☐ Use Defaults if Possible

☐ Generate Fields for Unresolved Relationships

Click on "Finish".

Persistence informations

Have a look to "Other Sources".
Open "META-INF".
What is this persistence.xml?



This file is generated by Netbeans to give informations about persistence, the database elements we just refered to.

Persistence informations

Click on button "Source".

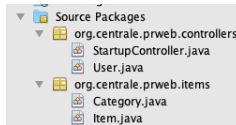
- It gives the entities we defined
- And all elements about the connection. Even your password...

Have a look to the tag "persistence-unit".
name is the name we will refer to.

Maybe we can rename it to **prwebSPRING_PU**. But your are not obliged to, and using the default name is ok.

Generated entities

Have a look to your package “items”.



Netbeans generated a class for each table we selected.

Generated entities

Have a look to item.java

```
@Entity
@Table(name = "item")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Item.findAll", query = "SELECT i FROM Item i"),
    @NamedQuery(name = "Item.findById", query = "SELECT i FROM Item i WHERE i.id = :id"),
    @NamedQuery(name = "Item.findByTitle", query = "SELECT i FROM Item i WHERE i.title = :title"),
    @NamedQuery(name = "Item.findByAuthor", query = "SELECT i FROM Item i WHERE i.author = :author"),
    @NamedQuery(name = "Item.findBody", query = "SELECT i FROM Item i WHERE i.body = :body")})
public class Item implements Serializable {
```

- Annotation **Entity** tells this is an Entity, an object we will manage and that refers to a table.
- Annotation **Table** gives the table name
- Annotation **NamedQueries** gives predefined requests for the persistence unit. Take care, it looks like SQL syntax but it is not. It is JPQL.

These annotations are “persistence” annotations.

Generated entities

Go down and have a look to the attributes.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Basic(optional = false)
@Column(name = "id")
private Integer id;
@Size(max = 255)
@Column(name = "title")
private String title;
@Size(max = 255)
@Column(name = "author")
private String author;
@Size(max = 255)
@Column(name = "body")
private String body;
@JoinColumn(name = "category_id", referencedColumnName = "id")
@ManyToOne
private Category categoryId;
```

Generated entities

- Attributes are private. We can access them through getters and setters.
- Annotation **Id** tells what is the id. GeneratedValue means there is an auto-increment.
- Annotation **Basic(optional = false)** tells it can't be null
- Annotation **Column** gives attribute name in the table
- Annotation **Size** gives String max size
- Annotation **JoinColumn** gives property name, and indicates it is linked to "id" in Category.
- Annotation **ManyToOne** tells 1 Item is linked to 1 Category, and 1 Category can be linked to many Items. That's why the Category is an object

Generated entities

Now have a look to Category.java

- First elements use same kind of annotations than Item
- itemCollection is a bit different: **OneToMany** tells 1 Category can be linked to many Items and 1 Item is linked to 1 Category. That's why itemCollection is a Collection of Items.

Managing entities

Create a package **org.centrale.prweb.repositories**.
Repositories are interfaces that manage objects. Each object in package items will have its repository.
Our interface will extend a predefined interface: JpaRepository

JpaRepository

JpaRepository is a SPRING interface. It also includes many annotations.

<https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>

This generic interface predefine CRUD actions and has its own implementations.

Managing entities

In package repositories, create a new interface **ItemRepository**.
This interface extends JpaRepository.

```
@Repository  
public interface ItemRepository extends JpaRepository<Item, Integer>
```

JpaRepository attributes:

- 1st attribute is the referenced entity, Item
- 2nd attribute is the entity id type, Integer

JpaRepository

JpaRepository is a Spring-data-jpa interface that uses AOP to implement many methods so that you don't have to do it. For example, save, delete, create, ... are already implemented and will generate SQL requests to the database.

ItemRepository is an interface, but Spring-data-jpa provides everything to make it operational. It implements classes and methods according to the Repository annotation and to the parameters you gave to JpaRepository.

A bit of configuration 1/5

Annotations will do their job, but we have to help them a little bit.

We told the application where are controllers, we will do the same for repositories.

We also have to explain we use JPA to manage object mapping to the database.

Modifications are all located in `applicationContext.xml`, after the controllers declaration and before `</beans>`

You will find the instructions in the materials in “`applicationContext v2.xml`”. Copy instructions to your `applicationContext.xml` file.

A bit of configuration 2/5

Here is what you have to change:

```
<jpa:repositories base-package="org.centrale.prweb.repositories"
                 entity-manager-factory-ref="entityManagerFactoryBean" />

<context:load-time-weaver/>
<bean id="entityManagerFactoryBean"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="persistenceUnitName" value="prwebSPRING_PU" />
  <property name="loadTimeWeaver">
    <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver" />
  </property>
</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactoryBean" />
</bean>
```

Let's analyse the content.

A bit of configuration 3/5

Like `context:component-scan`, **`jpa:repositories`** tells jpa where it can find repositories.

Each repository in that package will have a `@Repository` annotation.

A bit of configuration 4/5

Bean bean **id="entityManagerFactoryBean"** defines the EntityManager, the class that will manage entities and the link between objects and database tables, the persistence.

In that bean note the tag **property** that defines **persistenceUnitName**, the one defined in in persistence.xml.
Check it is yours.

The **loadTimeWeaver** property refers to the way we manage AOP, with classes linked to a database.

To use the loadTimeWeaver we have to load it, that's why there is a **context:load-time-weaver** instruction just before the previous tag.

A bit of configuration 5/5

At last, bean **id="transactionManager"** defines the way transactions are managed.
We let JPA do the job.

Managing auctions

We loaded the view “auctions”, but it is still a static page. We need:

- to load auctions from database
- transfer auctions to the view
- display auctions in the view

Loading auctions 1/5

When we connect, we have to display auctions.
That means the `handlePost` method in `StartupController.java` have to load auctions list and send it to the view.

To get the auctions list, we will ask the `ItemRepository`.
Let's create an `ItemRepository` in `StartupController`.

```
@Controller  
public class StartupController {  
    private ItemRepository itemRepository;
```

Loading auctions 2/5

Now, in `handlePost`, if connection is ok, let's get the auctions list.

```
if ((anUser.getUser().equals("admin")) && (anUser.getPasswd().equals("admin"))) {  
    List<Item> listItem = itemRepository.findAll();  
    returned = new ModelAndView("auctions");  
} else {
```

Remember “Clean and Build” ? Did you check PU name?

Let's check it's ok. Give Login, password.

No? “Error 500”? “Internal Server Error” because of a null pointer exception?

Why?

Loading auctions 3/5

ItemRepository is an interface, not a class. Where is its implementation?

More, we never created itemRepository. We said it is an ItemRepository, but we never allocated it.

That explains were the “null pointer exception” comes from.

So let's create a class that implements ItemRepository and create the itemRepository.

... no, we are joking.

Loading auctions 4/5

Remember JPA and AOP? There is no need of an implementation or an instantiation.

Let's add the annotation **Autowired**.

```
public class StartupController {  
    @Autowired  
    private ItemRepository itemRepository;  
}
```

Loading auctions 5/5

Let's check it's ok.

Add a breakpoint in handlePost.

Run project in debug mode. Check you've got the auctions in "listItem".

```
ModelAndView returned;
if ((anUser.getUser().equals("admin")) && (anUser.getPass() != null)) {
    List<Item> listItem = itemRepository.findAll();
    returned = new ModelAndView("auctions");
} else {
```

centrale.prwebspring.controllers.StartupController > handlePost > if ((anUser

Name	
<Enter new watch>	
this	StartupController
anUser	User
listItem	List<Item>
[0]	Item
[1]	Item

Why does it work?

Annotation Autowired takes next line as an implementation constraint and manages it.

Autowired creates the implementation of classes, methods, ... And as it uses AOP, it builds it according to ItemRepository and to Item.

Remember "NamedQuery" in Item?

Have a look to the queries names. Have a look to the method we used in handlePost. See something similar?

That is the result of "Repository", "Autowired", "NamedQueries", ...
That is AOP.

Custom repositories

Maybe JpaRepository methods will not be sufficient for you.
You can add your own methods using custom repositories.

Change Repository declaration :

```
@Repository  
public interface ItemRepository extends JpaRepository<Item, Integer>  
    , ItemCustomRepository {  
}
```

Custom repositories

Then we create the required elements

First, the interface :

```
public interface ItemCustomRepository {  
    // Add your methods declarations here  
}
```

Custom repositories

Then, the way we implement it :

```
@Repository
public class ItemCustomRepositoryImpl implements ItemCustomRepository {
    @Autowired
    @Lazy
    ItemRepository itemRepository;
    // Add your methods definitions here
}
```

Custom queries

You can define methods in the interface using standart queries.
Here are 3 examples.

First one uses JPQL. Next one is a native SQL example. Last one uses a parameter.

YOU WILL NOT NEED THESE EXAMPLES FOR THE PROJECT

```
public interface ItemRepository extends JpaRepository<Item, Integer>, ItemCustomRepository {  
  
    @Query("SELECT i FROM Item i WHERE i.categoryId IS NULL")  
    Collection<Item> findAllWithoutCategory();  
  
    @Query(value="SELECT i FROM Item i WHERE i.Category_Id IS NULL", nativeQuery=true)  
    Collection<Item> findAllNativeWithoutCategory();  
  
    @Query("SELECT i FROM Item i WHERE i.author=:author")  
    Collection<Item> findWithParameter(@Param("author")String author);  
  
}
```

Send data to the view.

Next step consists in sending data to the view.

In ModelAndView, there is a method “addObject” that adds an object to the view.

Let’s use it.

```
List<Item> listItem = itemRepository.findAll();  
returned = new ModelAndView("auctions");  
returned.addObject("listItems", listItem);
```

In our code, addObject creates a “view object” called “listItems” that contains our object listItem.

You can use addObject for any object you have to send to the view.

JSTL

JSTL is the **J**ava server page **S**tantard **T**ag **L**ibrary. It helps developer for jsp pages.

You can use:

- objects added to the view using `${objectName}`
- attributes in an object using `${objectName.attributeName}`
Take care: **objectName.attributeName** is translated to **objectName.getAttributeName()**
- use methods on objects `${objectName.method(attributes)}`
- tags like `if` -to test something-, `forEach` -to loop on an iterable objects-, ...
- request and response as predefined objects
- and more

JSTL - forEach

forEach is used to loop on an iterable object

```
<c:forEach var="..." items="..." >  
...  
</c:forEach>
```

- **var** set the loop variable name
- **items** is an iterable object

JSTL - if

if is used to check values

```
<c:if test="..." >  
...  
</c:if>
```

- **test** is the test to perform

There is no "else" available.

JSTL - choose

choose is used to select instruction according to values

```
<c:choose >  
<c:when test="..." > ... </c:when>  
...  
<c:otherwise> ... </c:otherwise>  
</c:if>
```

- **test** is the test to perform

JSTL examples

- Test example

```
<c:if test="${user=='admin'}">Manager</c:if>
```

- Choose example

```
<c:choose>  
  <c:when test="${user=='admin'}">Manager</c:when>  
  <c:when test="${user=='root'}">Administrator</c:when>  
  <c:otherwise>${user}</c:otherwise>  
</c:choose>
```

JSTL expression evaluationL.

- Remember expressions are evaluated through `${...}`
- Logical expressions
 - `&&` is and
 - `||` is or
 - `!` is not
 - `==` is equal to
 - `!=` is different from
- **empty** can be applied to any kind of object to check if it is null or empty.
 - `${empty user}` is equivalent to `${ (user == null) || (user.equals("")) }`
 - `${empty list}` is equivalent to `${ (list == null) || (list.isEmpty()) }`
 - ...

Using an object in the view 1/2

In our view, we will use the objects sent by the view through JSTL.

Open the auctions view page in Netbeans.

Add this taglib line to your jsp file to include JSTL core library.

Functionalities will be prefixed by "c:".

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
  <head>
    <title>Auctions Page</title>
```

Using an object in the view 2/2

Now, let's display the list of items. StartupController gave us the object "listItems".

We have to loop on this array and display its content.

```
<c:forEach var="item" items="${listItems}">
  <tr>
    <td>${item.id}</td>
    <td>${item.title}</td>
    <td>${item.author}</td>
    <td>${item.body}</td>
    <td></td>
    <td style="text-align:center"><button></button></td>
  </tr>
</c:forEach>
```

Our list of auctions

And the result -depending on your database content-:

List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JY Martin	I sell my old computer. 3 years old		
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)		
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?		
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?		
					

Removing an auction 1/4

We can list the auctions. Now, we have to add some actions on our objects. We start with removing auctions.

Add a form around your delete button. Your form calls delete.do using POST method.

It also needs the auction's id to delete the auction.

```
<td style="text-align:center">  
  <form action="delete.do" method="POST">  
    <input type="hidden" name="id" value="${item.id}" />  
    <button></button>  
  </form>  
</td>
```


Removing an auction 2/4

Create a new controller `ItemsController.java`

- We need an `ItemRepository`
- Route is "delete.do"
- GET method should lead to view index.
- POST method uses a `HttpServletRequest` as a parameter
 - Get parameter "id" from the `HttpServletRequest` as an Integer
 - Use `findById` from your `ItemRepository` to get the Item (*)
 - if it is present, delete it from your `ItemRepository` with "delete"
 - in any case, use `auctions` as the returned view. Do not forget the list of auctions.

(*)`findById` returns an `Optional<Item>`.

<https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>

Removing an auction 3/4






Your script to delete the auction should look like this.

```
String idStr = request.getParameter("id");  
if (idStr != null) {  
    int id = Integer.parseInt(idStr);  
    Optional<Item> item = itemRepository.findById(id);  
    if (item.isPresent()) {  
        itemRepository.delete(item.get());  
    }  
}
```

Removing an auction 4/4

Clean and Rebuild your project, run it.
Delete an item.
Check it is also deleted in your database.

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)		
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?		
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?		
					

Adding an auction 1/2

Now let's add auctions.

Add a form around the jsp line that should add an auction.
It would be a good idea to include the full TR line. Take care to inputs names. They should be the same as Item attributes.

- form action is add.do, in POST mode.
- add a route in ItemsController.
- Use a ModelAttribute with Item to get an Item.
- Use save method from ItemRepository to create the Item in database.

Adding an auction 2/2

Clean and Rebuild your project, run it. Add an item.
Check it is also added in your database.

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)		
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?		
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?		
	Anime	JY Martin	Looking for full version of Daimachi		

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)		
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?		
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?		
7	Anime	JY Martin	Looking for full version of Daimachi		
					

What next? Categories.

Currently, we do not manage categories.

- Create a repository for Categories and use JpaRepository to manage it.
- Display categories in the auctions view.
Remember, `x.y` means `x.getY()`. So you should be able to get the item's category, and the category's name.

Check categories are displayed. Maybe something can go wrong. Because of a null category. Maybe you can use JSTL choose for that.

What next? Categories.

If you have problem displaying categories, you can do something like that:

```
<c:choose>
  <c:when test="{empty item.categoryId}">--</c:when>
  <c:otherwise>${item.categoryId.name}</c:otherwise>
</c:choose>
```

What next? Categories.

You might have something like this:

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for full version of Daimachi	-	
					

Selecting a category when adding

Also, we have to select categories when adding an auction.

- Give the categories list to the view. Use a CategoryRepository for that.
- In the auction view, create a SELECT tag in the right TD, when adding an item
 - values are the categories values
 - text displayed is the name of the category
 - SELECT name is ...
Let's talk about that

Selecting a category when adding

According to AOP, and so on, it would be nice to set the SELECT name to "categoryID" so that AOP retrieves category and creates an item with the right category. Unfortunately, it doesn't work.

Creating a ModelAttribute for category in ItemsAddController doesn't either.







Solution:

- Set SELECT name to "category" and use a HttpServletRequest in the controller.
- Retrieve category value, and get Category.
- Set auction category to that one.
- Save auction.

Selecting a category when adding

Here is the result :

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for full version of Daimachi	-	
	Anime	JY Martin	Selling LODOSS war	electronics	

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for full version of Daimachi	-	
8	Anime	JY Martin	Selling LODOSS war	electronics	
				electronics	

AJAX calls

AJAX calls are quite easy to implement too.

- An AJAX call is a standart call, with parameters.
- So you have to implement a controller that manages your AJAX call.
- This controller uses a specific view that returns a JSON object.

You will need the JSON library to manage JSON items. In your dependencies, search for "org.json" and retrieve the JAR library from "org.json : json".

AJAX calls

JSON library implements several object you can use:

- JSONObject: a JSON object.
- JSONArray: a JSON Array

You can “put” json objects, json arrays, strings, integers, ... to JSONObject. You can “put” json objects, json arrays, strings, integers, ... in a JSONArray.

Avoid methods like append or accumulate that may create arrays where there should not be.

AJAX calls

In the materials, you can find an ajax view you can use for ajax exchanges.

Here is an example of the way you can use it.

Script

```
public ModelAndView handlePost() {  
    JSONObject returnedObject = new JSONObject();  
    returnedObject.put("id", 1);  
    JSONArray anArray = new JSONArray();  
    anArray.put("abc");  
    anArray.put("def");  
    returnedObject.put("myArray", anArray);  
  
    ModelAndView returnedValue = new ModelAndView("ajax");  
    returnedValue.addObject("theResponse", returnedObject.toString());  
    return returnedValue;  
}
```

JSON result

```
{  
  "myArray": [  
    "abc",  
    "def"  
  ],  
  "id": 1  
}
```

Updating auctions

We will add an "Edit" button that will switch texts to inputs.
"Edit" button will be replaced by a "Save" button.
User will change informations.
At last, a click on the "Save" button saves informations and switch
"Save" button to "Edit" button.

Updating auctions

Managing “Edit” button is quite simple. We use AJAX to get informations and we switch buttons.

For the “Save” button, we can use one of the 2 following methods:

- We continue with AJAX.
Save sends informations through AJAX and receive an information about completion. It switches back to “Edit”
- We use a form. We put informations in an hidden form. We call a route -and a controller- that saves informations and reload the page.

You can use the one you prefer.

Updating auctions -both methods-

In the auction view:

- include jQuery JS file
- create a file auctions.js and include it
- Add an "edit" button for each auction.
on click launch an editAuction function. That function will be in auctions.js
- Add an "save" button for each auction, and set it as not displayed. on click launch a saveAuction function. That function will be in auctions.js

Updating auctions -both methods-

in auctions.js

- create a editAuction function.
 - Maybe having id as a parameter would be a good idea.
 - using AJAX, get objet's attributes. Call route "getItem.do"
 - replace TD contents
 - ID becomes a hidden INPUT item.
 - author, body and title become INPUT items.
 - Category becomes a SELECT item with the categories.
 - hide edit button, show save button

Updating auctions -both methods-

In your controller -or another one if you choose to create a new controller to manage AJAX-

- Manage route "getItem.do"
- method GET should return an empty JSON element
- method POST method should
 - get the ID
 - find corresponding Item
 - return a JSON element with
 - all attributes of the corresponding object
 - the list of available categories

Maybe scripts next 2 slides may help you.

Updating auctions -both methods-

Creating JSON object

```
JSONObject returnedObject = new JSONObject();
returnedObject.put("id", item.getId());
returnedObject.put("title", item.getTitle());
returnedObject.put("body", item.getBody());
returnedObject.put("author", item.getAuthor());
if (item.getCategoryId() != null) {
    returnedObject.put("category", item.getCategoryId().getId());
    returnedObject.put("categoryName", item.getCategoryId().getName());
} else {
    returnedObject.put("category", -1);
    returnedObject.put("categoryName", "");
}
```

Updating auctions -both methods-

Creating JSON array

```
List<Category> listCategory = categoryRepository.findAll();
JSONArray categories = new JSONArray();
for (Category category : listCategory) {
    JSONObject jsonCategory = new JSONObject();
    jsonCategory.accumulate("id", category.getId());
    jsonCategory.accumulate("name", category.getName());
    categories.put(jsonCategory);
}
returnedObj.put("categories", categories);
```

Updating auctions -1rst method-

in auctions.js

- create a saveAuction function.
 - get informations from inputs and select
 - use AJAX, save object. You will have to create a route for that
 - replace TD contents -inputs and select values- by texts in the line
 - hide edit button, show save button
- create a route in your controller to save informations. Return an acknowledgment.

Updating auctions -2nd method-

- add a form in your jsp file
- add hidden field for each of the fields - 1 hidden field per information to save-

in auctions.js

- create a saveAuction function.
 - get informations from inputs and select
 - copy informations from input fields to the hidden fields of the form
 - submit form

Updating auctions -2nd method-

- create a route in your controller
- get the item, update informations and save
- load informations like you did to display the page
- send page.

