Spring

Controller y View

Spring: Controller

- Un Controller es una clase que organiza métodos llamados handlers.
- Como siempre toda clase debe ser nombrada en CamelCase.
- Y adicionalmente Spring solicita que el nombre de la clase termine con el sufijo “Controller”.
- También solicita que la clase sea “decorada”, o que tenga la anotación *@Controller*

```
@Controller
public class SiteController{

}
```

Spring: Controller Mapping

- Para que Spring sepa que todos handlers de un controller están organizados bajo un mismo nombre, debemos definir esto explícitamente.
- Podemos usar las anotaciones:
  - RequestMapping
    - @RequestMapping(“/compras”)

```
@Controller
@RequestMapping(“/site”)
public class SiteController{

}
```

Spring: Handlers

- Cada método, dentro de un controller, debe retornar un String.
- Y el String retornado puede resultar en la búsqueda de un template o en una re dirección a otro controller.

```
@Controller
@RequestMapping(“/site”)
public class SiteController{

    public String home(Model model){
        return “index”;
    }
}
```

Spring: Handlers Mapping

- Para que Spring sepa exactamente que handlers debe atender las peticiones URL hacia nuestra aplicación, debemos definir lo explícitamente.
- Podemos usar las anotaciones:
  - RequestMapping
  - MethodMapping:
    - GetMapping
    - PostMapping
    - …

```
@Controller
@RequestMapping(“/site”)
public class SiteController{

    @GetMapping(“/home”)
    public String home(Model model){
        return “index”;
    }
}
```

Spring: View

- Cada vez que un handler retorna un String se buscará un archivo con el nombre que haga match con el String retornado.
- Comúnmente será un archivo html.
- Se recomiende que el charset del html sea el mismo que el del proyecto. Para hacerlo el proyecto lo mas portable posible use UTF-8.

Spring: Handlers Parameters to View

- Para el envío de datos desde el handler a la vista, tenemos diferentes métodos.
  - Model
    - addAtribute(“key”,value)
  - ModelMap
    - addAtribute(“key”,value)
  - Map<String,Objetc>
    - Put(“key”,value)
  - ModelAndView (Esto cambia la estructura de retorno)
    - AddObject(Object, Object)
    - setViewName()

```
@Controller
@RequestMapping(“/site”)
public class SiteController{

    @GetMapping(“/home”)
    public String home(Model model){
        model.addAttribute(“title”, ”Home”);
        return “index”;
    }
}
```

Spring: View Parameters

- Para la utilización de los datos enviados desde el handler hacia la vista, usamos a Thymeleaf.
- Thymeleaf
  - Motor de interpretación de plantillas.
  - Acceso a los datos:
    - ${}, acceso a los datos de la plantilla
    - @{}, acceso a handlers, recursos estáticos (rutas URL)

```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf
.org">
<head>
<meta charset="UTF-8">
<title th:text=”${title}”>
</title>
</head>
<body>
<h1> </h1>
</body>
</html>
```

Spring: Handlers Parameters to Controller

- Para el envío de datos desde la vista a un controller, tenemos el métodos.
  - @RequestParam(defaultValue String, name String, required boolean)
  - @PathVariable()
  - Ejemplo URL:

http://localhost:8080/home?title=Paso%20de%20Parametros

```
@Controller
@RequestMapping(“/site”)
public class SiteController{

    @GetMapping(“/home”)
    public String
home(@RequestParam String
title, Model model){
        model.addAttribute(“title”,
title);
        return “index”;
    }
}
```

Spring: Handlers Parameters to Controller

- Para el envío de datos desde la vista a un controller, tenemos el métodos.
  - @RequestParam(defaultValue String, name String, required boolean)
  - @PathVariable(defaultValue String, name String, required boolea)

http://localhost:8080/home/Paso%20de%20Parametros

```
@Controller
@RequestMapping(“/site”)
public class SiteController{

  @GetMapping(“/home/{title}”)
    public String
home(@PathVariable String
title, Model model){
        model.addAttribute(“title”,
title);
        return “index”;
    }
}
```