Tema 3. Tecnologías del servidor. Presentación: Spring MVC y Thymeleaf

Programación web

Boni García

Curso 2017/2018

Índice

1. Introducción: Java en el lado servidor
2. Presentación: Spring MVC y Thymeleaf
3. Acceso a bases de datos: Spring Data JPA
4. Seguridad: Spring Security

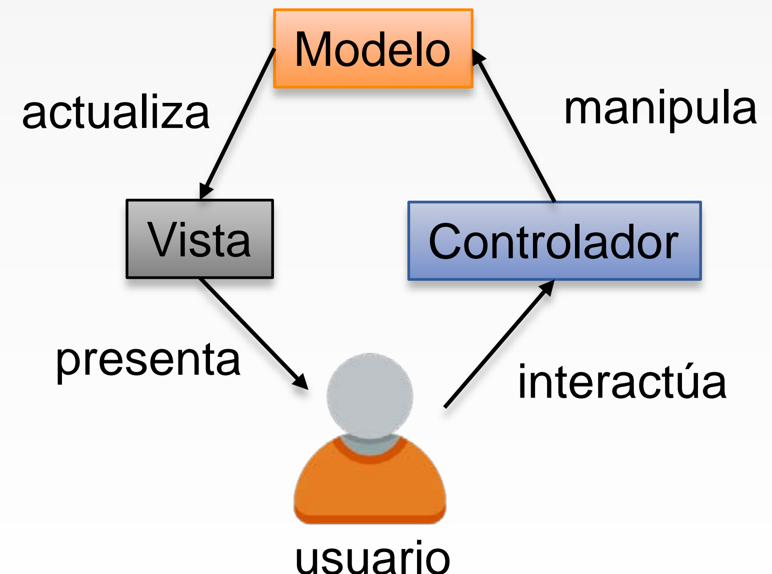
Índice

1. Introducción: Java en el lado servidor
2. Presentación: Spring MVC y Thymeleaf
   - Spring MVC
   - Thymeleaf
3. Acceso a bases de datos: Spring Data JPA
4. Seguridad: Spring Security

Spring MVC

Introducción

- El modelo vista controlador (**MVC**) es un patrón de diseño que se usa en aplicaciones con interfaz gráfica de usuario (GUI, *Graphical User Interface*) para la desacoplar la interfaz de usuario de la lógica de negocio
- En MVC se manejan tres componentes:
  - **Controlador**: Responde a eventos (acciones de usuario) y genera el modelo
  - **Modelo**: Es la representación de la información que maneja el sistema
  - **Vista**: Presenta el modelo en un formato adecuado a la GUI



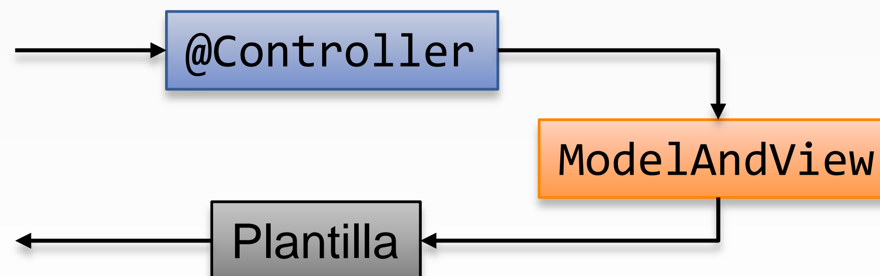
Spring MVC

MVC en Spring

- En Spring los controladores (*Controllers*) son un tipo de componente (vean) encargados de atender las peticiones web
  - Procesan los datos que llegan en la petición (parámetros)
  - Hacen peticiones a la base de datos, usan diversos servicios, etc…
  - Definen la información que será visualizada en la página web (el modelo)
  - Determinan que vista será la encargada de generar la página HTML

Petición desde el navegador (URL)

Página HTML



http://spring.io/guides/gs/serving-web-content/

Spring MVC

MVC en Spring

- Las vistas en Spring MVC se implementan con tecnologías de plantillas
- Generan HTML partiendo de una plantilla y la información que viene del controlador (modelo)
- Diferente tecnologías de plantillas se pueden usar con Spring MVC:
  - JSP: http://www.oracle.com/technetwork/java/javaee/jsp/
  - FreeMarker: http://freemarker.org/
  - Velocity: http://velocity.apache.org/
  - Mustache: https://mustache.github.io/
  - Thymeleaf: http://www.thymeleaf.org/

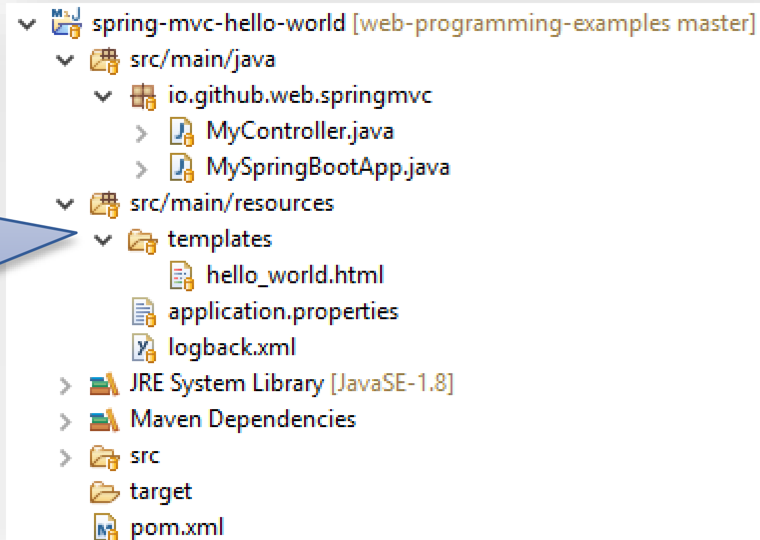
Las plantillas Thymeleaf pueden visualizarse en un navegador sin necesidad de ser procesadas (*natural templating*)

Spring MVC

Fork me on GitHub

Ejemplo

- Estructura de la aplicación vista desde Eclipse:



Por convención, las plantillas están alojados en la carpeta `templates`

Spring MVC

Ejemplo

pom.xml

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

Declaramos dos dependencias en nuestro pom.xml:

- `spring-boot-starter-web`: (por ser una app web)
- `spring-boot-starter-thymeleaf`: (por usar Thymeleaf como motor de plantillas)

Spring MVC

Ejemplo

Declaramos el mapeo petición-controlador mediante @RequestMapping (por defecto para peticiones GET)

El método devuelve una instancia de ModelAndView, que es el modelo de datos que le llega a la plantilla

MyController.java

```
package io.github.web.springmvc;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class MyController {

    @RequestMapping("/")
    public ModelAndView greeting() {
        ModelAndView model = new ModelAndView("hello_world");
        model.addObject("name", "World");
        return model;
    }
}
```

Elegimos el nombre de la plantilla en el constructor de ModelAndView

Spring MVC

Ejemplo

- Vistas (implementado con **Thymeleaf**)

hello_world.html

```
<!DOCTYPE html>
<html>
<body>
   <h1>Spring MVC with Thymeleaf</h1>
   <p>
      Hello <span data-th-text="${name}"></span>!
   </p>
</body>
</html>
```

Desde la versión 3 de Thymelaf se usan atributos HTML5 del tipo `data-*` para gestionar las plantillas

El acceso a datos del modelo en modo texto se hace usando `data-th-text`

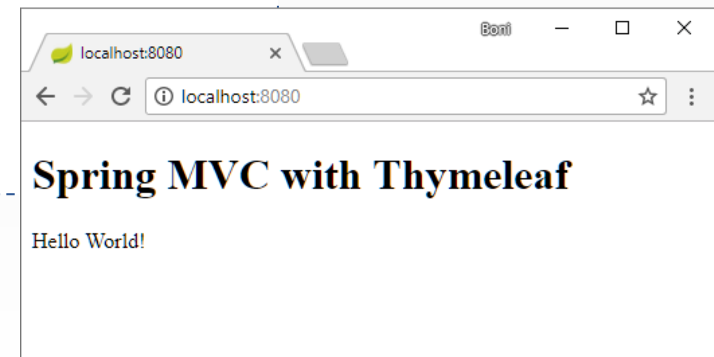
Spring MVC

Ejemplo

- Como siempre ocurre en Spring Boot, nuestra aplicación se ejecuta como una aplicación Java normal

MySpringBootApp.java

```
package io.github.web.springmvc;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MySpringBootApp {

    public static void main(String[] args) {
        SpringApplication.run(MySpringBootApp.class, args);
    }
}
```



Thymeleaf

Introducción

- La sintaxis de las plantillas Thymeleaf se define en páginas HTML5 mediante atributos `data-th-*` dentro de etiquetas HTML

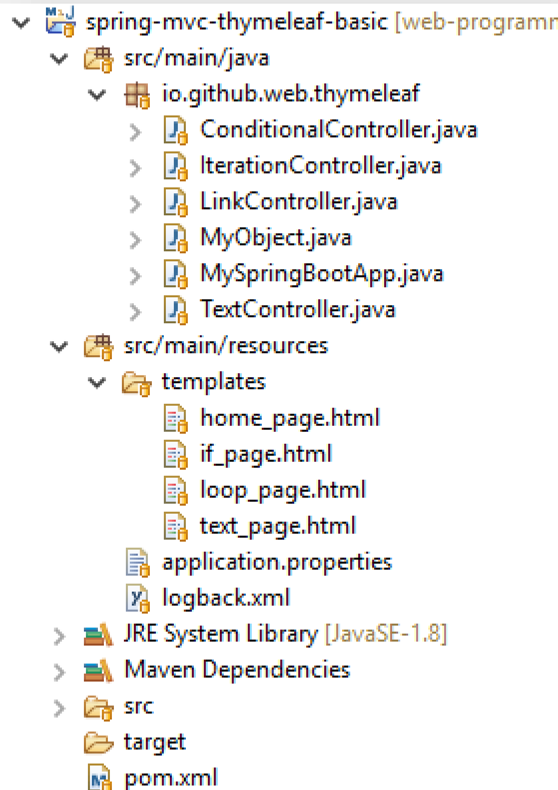
```
<!DOCTYPE html>
<html>
<body>
   <p>Hello <span data-th-text="${name}"></span></p>
</body>
</html>
```

Usaremos la sintaxis `${…}` para acceder a los objetos del modelo desde las plantillas

http://www.thymeleaf.org/documentation.html

Thymeleaf

Atributos básicos



pom.xml

```xml
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.0.0.RELEASE</version>
</parent>

<properties>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <java.version>1.8</java.version>
</properties>

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

Fork me on GitHub

Thymeleaf

Atributos básicos

- Los datos en formato texto se manejan en Thymeleaf con el atributo `data-th-text`

```
public class MyObject {

    private String name;
    private String description;

    public MyObject(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String sayHello() {
        return "Hello!!!";
    }

    // Getters, setters, and toString
}
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class TextController {

    private final Logger log = LoggerFactory.getLogger(this.getClass());

    @RequestMapping("/text")
    public ModelAndView processText() {
        log.info("GET /text");

        ModelAndView model = new ModelAndView("text_page");
        model.addObject("greetings", "Hello world!");

        MyObject myObject = new MyObject("My name", "My description");
        model.addObject("myobj", myObject);

        log.info("Model: {}", model);
        return model;
    }

}
```

Thymeleaf

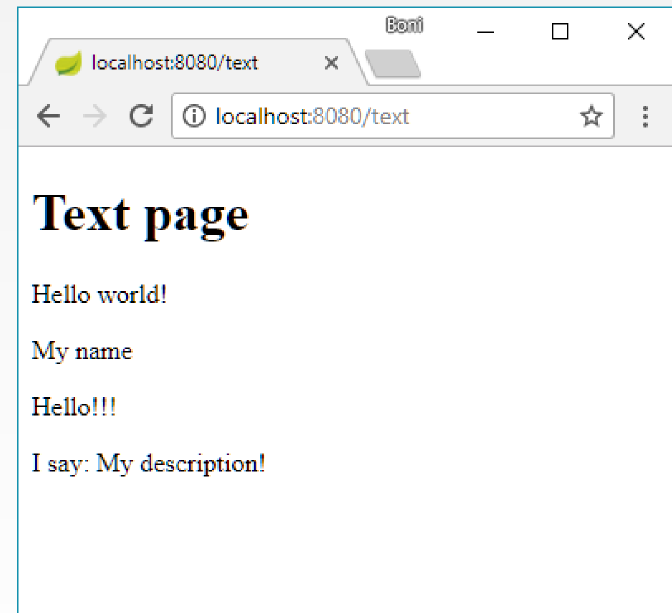
Atributos básicos

- Los datos en formato texto se manejan en Thymeleaf con el atributo `data-th-text`

text_page.html

```
<!DOCTYPE html>
<html>
<body>
    <h1>Text page</h1>
    <p data-th-text="${greetings}"></p>
    <p data-th-text="${myobj.name}"></p>
    <p data-th-text="${myobj.sayHello()}"></p>
    <p data-th-text="|I say: ${myobj.description}!|"></p>
</body>
</html>
```

Accedemos a los datos del modelo desde la plantilla



Thymeleaf

Atributos básicos

- Podemos condicionar la visualización de elemento en la plantilla mediante `data-th-if`

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class ConditionalController {

    private final Logger log = LoggerFactory.getLogger(this.getClass());

    @RequestMapping("/if")
    public ModelAndView processIf() {
        log.info("GET /if");

        ModelAndView model = new ModelAndView("if_page");
        model.addObject("visible", true);
        model.addObject("hidden", false);

        log.info("Model: {}", model);
        return model;
    }

}
```

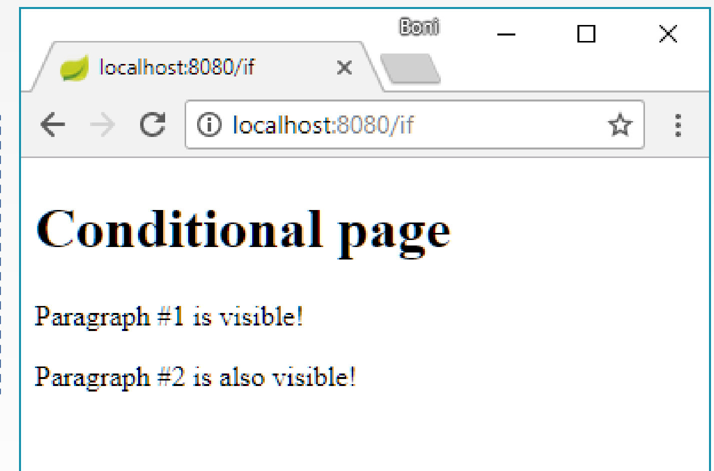
Thymeleaf

Atributos básicos

- Podemos condicionar la visualización de elemento en la plantilla mediante `data-th-if`

if_page.html

```
<!DOCTYPE html>
<html>
<body>
    <h1>Conditional page</h1>
    <p data-th-if="${visible}">Paragraph #1 is visible!</p>
    <p data-th-if="${not hidden}">Paragraph #2 is also visible!</p>
</body>
</html>
```



Thymeleaf

Atributos básicos

- Podemos recorrer colecciones mediante `data-th-each`

```java
import java.util.ArrayList;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class IterationController {

    private final Logger log = LoggerFactory.getLogger(this.getClass());

    @RequestMapping("/loop")
    public ModelAndView processLoop() {
        log.info("GET /loop");

        ModelAndView model = new ModelAndView("loop_page");
        List<String> colors = new ArrayList<>();
        colors.add("red");
        colors.add("blue");
        colors.add("green");
        model.addObject("colors", colors);

        log.info("Model: {}", model);
        return model;
    }

}
```

Thymeleaf

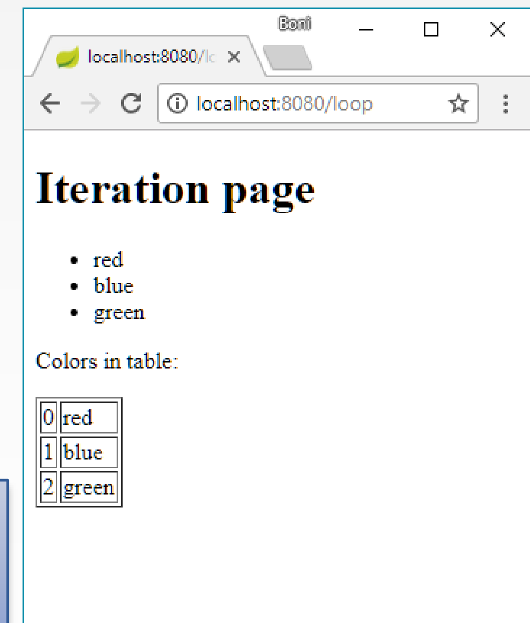
Atributos básicos

- Podemos recorrer colecciones mediante `data-th-each`

loop_page.html

```
<!DOCTYPE html>
<html>
<body>
    <h1>Iteration page</h1>
    <ul>
        <li data-th-each="color : ${colors}" data-th-text="${color}">Color</li>
    </ul>

    <p>Colors in table:</p>
    <table border="1">
        <tr data-th-each="color, it : ${colors}">
            <td data-th-text="${it.index}"></td>
            <td data-th-text="${color}">Color</td>
        </tr>
    </table>
</body>
</html>
```

Se puede declarar una variable adicional para guardar información de la iteración



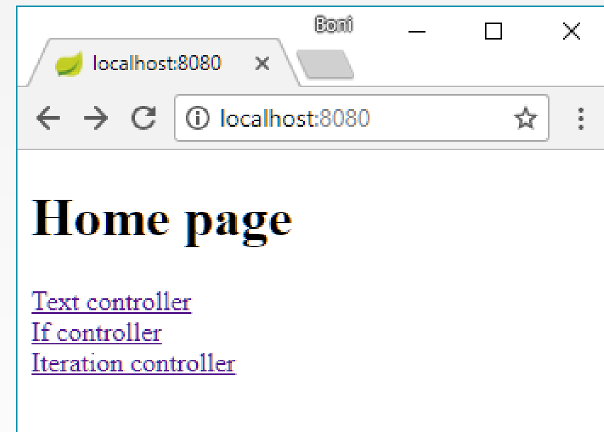
Thymeleaf

Atributos básicos

- Los enlaces en las plantillas Thymeleaf los generamos mediante el atributo `data-th-href`

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class LinkController {

    @RequestMapping("/")
    public ModelAndView processHome() {
        ModelAndView model = new ModelAndView("home_page");
        return model;
    }

}
```

```html
<!DOCTYPE html>
<html>
<body>
    <h1>Home page</h1>
    <a data-th-href="@{/text}">Text controller</a><br>
    <a data-th-href="@{/if}">If controller</a><br>
    <a data-th-href="@{/loop}">Iteration controller</a><br>
</body>
</html>
```



Otros atributos equivalentes son `data-th-src` (para enlaces a scripts) y `data-th-action` (para la dirección de envío de formularios)

Thymeleaf

Fork me on GitHub

Envío y recepción de datos

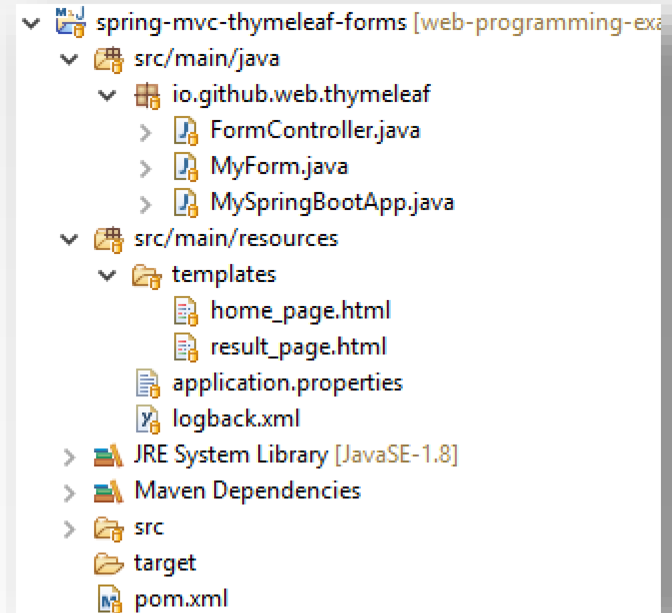
- Los datos individuales los recibimos en el lado servidor usando la anotación `@RequestParam`

home_page.html

```
<h2>Form 1</h2>
<form data-th-action="@{processForm1}" method="post">
    <label>One input fields</label><br>
    <input type="text" name="input"><br>
    <input type="submit">
</form>
```

FormController.java

```
@RequestMapping("/processForm1")
public ModelAndView processForm1(@RequestParam String input) {
    ModelAndView model = new ModelAndView("result_page");
    model.addObject("result", input);
    return model;
}
```



Esta misma técnica nos sirve para recibir datos enviados por URL (por ejemplo http://my-server.com/path?option=web&view=category&lang=es)

Thymeleaf

Fork me on GitHub

Envío y recepción de datos

- Cuando los datos que se manejan son complejos, se pueden gestionar usando una clase Java y `@ModelAttribute`

home_page.html

```html
<h2>Form 2</h2>
<form data-th-action="@{processForm2}" method="post">
    <label>Several input fields</label><br>
    <input type="text" name="info1"><br>
    <input type="text" name="info2"><br>
    <input type="submit">
</form>
```

FormController.java

```java
    @RequestMapping("/processForm2")
    public ModelAndView processForm2(@ModelAttribute MyForm info) {
        ModelAndView model = new ModelAndView("result_page");
        model.addObject("result", info.toString());
        return model;
    }
}
```

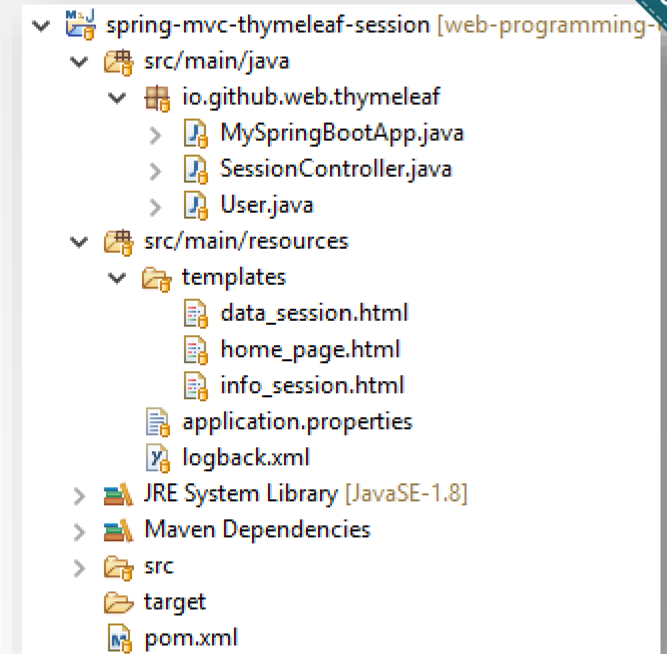
MyForm.java

```java
public class MyForm {

    private String info1;
    private String info2;

    // Getters, setters, and toString()

}
```

Thymeleaf

Gestión de datos de sesión

- Es habitual que las aplicaciones web gestionen información diferente para cada usuario que está navegando. Por ejemplo:
  - Amigos en Facebook
  - Lista de correos en Gmail
  - Carrito de la compra en Amazon
- Podemos gestionar esta información de manera muy sencilla con Spring



Thymeleaf

Gestión de datos de sesión

- Crearemos un componente Spring (`@Component`) especial que se asociará a cada usuario

```
import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.stereotype.Component;
import org.springframework.web.context.WebApplicationContext;

@Component
@Scope(value = WebApplicationContext.SCOPE_SESSION, proxyMode = ScopedProxyMode.TARGET_CLASS)
public class User {

    private String info;

    public void setInfo(String info) {
        this.info = info;
    }

    public String getInfo() {
        return info;
    }
}
```

Thymeleaf

Gestión de datos de sesión

- Usaremos inyección de dependencias para usar es componente en los controladores
- En este ejemplo, el objeto `user` tendrá diferentes valores para los diferentes usuarios, mientras que el objeto `sharedInfo` será común para todos

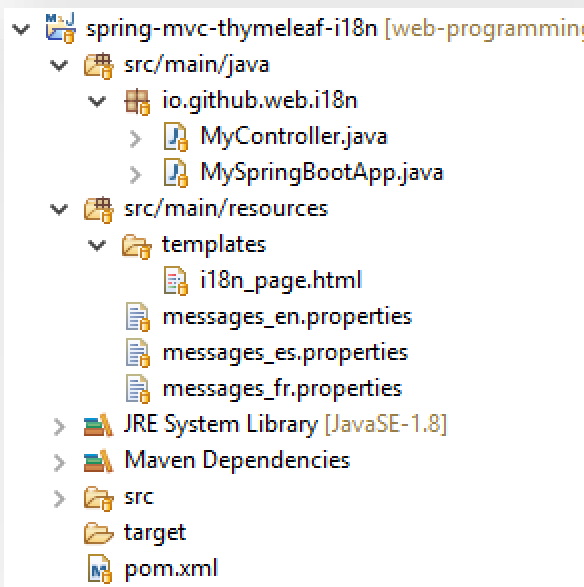
```
@Controller
public class SessionController {

    private String sharedInfo;
    private User user;

    public SessionController(User user) {
        this.user = user;
    }

    @RequestMapping(value = "/")
    public ModelAndView processForm() {
        return new ModelAndView("home_page");
    }

    @RequestMapping(value = "/processFormSession")
    public ModelAndView processForm(@RequestParam String info) {
        user.setInfo(info);
        sharedInfo = info;
        return new ModelAndView("info_session");
    }

    @RequestMapping("/showDataSession")
    public ModelAndView showData() {
        String userInfo = user.getInfo();
        return new ModelAndView("data_session").addObject("userInfo", userInfo)
                .addObject("sharedInfo", sharedInfo);
    }
}
```

Thymeleaf

Fork me on GitHub

Soporte de internacionalización (I18N)

- Spring MVC con Thymeleaf tiene soporte nativo para aplicaciones multi-idioma (internacionalización, I18N)



messages_en.properties

```
welcome=Welcome to my web!
```

messages_es.properties

```
welcome=Bienvenido a mi web!
```

messages_fr.properties

```
welcome=Bienvenue sur mon site web!
```

Thymeleaf

Soporte de internacionalización (I18N)

MySpringBootApp.java

```
@SpringBootApplication
public class MySpringBootApp implements WebMvcConfigurer {
  @Bean
  public MessageSource messageSource() {
    ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
    messageSource.setBasename("messages");
    return messageSource;
  }
  @Bean
  public LocaleResolver localeResolver() {
    SessionLocaleResolver sessionLocaleResolver = new SessionLocaleResolver();
    sessionLocaleResolver.setDefaultLocale(Locale.ENGLISH);
    return sessionLocaleResolver;
  }
  @Bean
  public LocaleChangeInterceptor localeChangeInterceptor() {
    LocaleChangeInterceptor result = new LocaleChangeInterceptor();
    result.setParamName("lang");
    return result;
  }
  @Override
  public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(localeChangeInterceptor());
  }
}
```

Nombre del fichero que contiene los mensajes de I18N → messageSource

Región (locale) por defecto → localeResolver

Parámetro con el que cambiar la región desde URL → localeChangeInterceptor

Thymeleaf

Soporte de internacionalización (I18N)

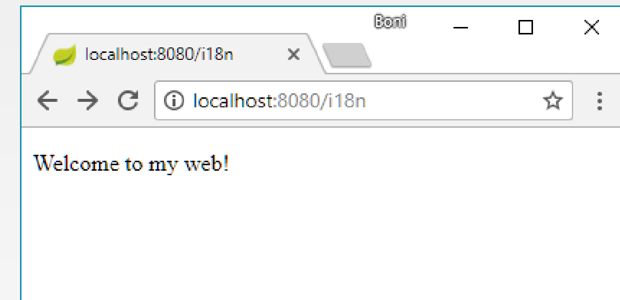
MyController.java

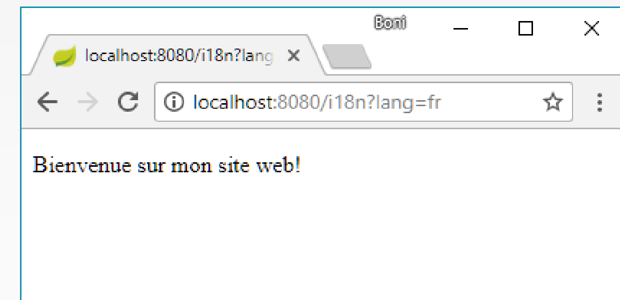
```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class MyController {

    @RequestMapping("/i18n")
    public ModelAndView i18n() {
        return new ModelAndView("i18n_page");
    }

}
```

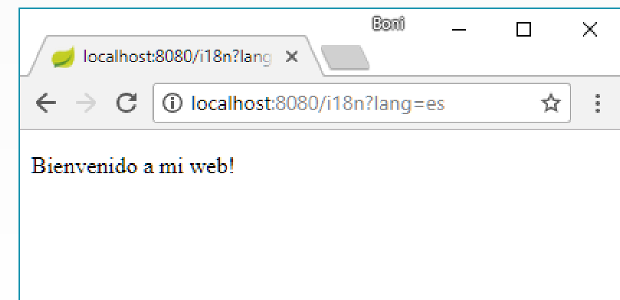
i18n_page.html

```
<!DOCTYPE html>
<html>
<body>
   <p data-th-text="#{welcome}"></p>
</body>
</html>
```

El símbolo # hace que la plantilla se rellene con los datos de I18N



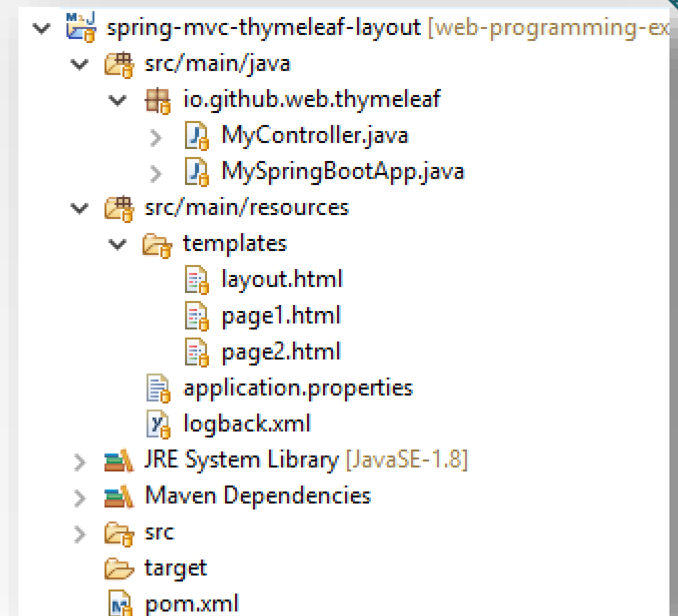




Thymeleaf

Layouts

- Normalmente las aplicaciones web tienen partes comunes (cabecera, pie de página, menú, …)
- Thymeleaf proporciona mecanismos para realizar la disposición de los elementos de una página (*layout*) para evitar la duplicidad de estructura y código en las páginas web



http://www.thymeleaf.org/doc/articles/layouts.html

Thymeleaf

Layouts

```
@Controller
public class MyController {

  @RequestMapping(value = "/page1")
    public ModelAndView layout1() {
        return new ModelAndView("page1");
    }

    @RequestMapping(value = "/page2")
    public ModelAndView layout2() {
        return new ModelAndView("page2");
    }

}
```

layout.html

```
<!DOCTYPE html>
<html>
<body>
    <div class="container">
        <div class="header">
            <h1>Header</h1>
        </div>
        <div class="content">
            <div class="nav">
                Section 1<br> Section 2<br> Section 3
            </div>
            <div class="main" data-layout-fragment="content"></div>
        </div>
        <div class="footer">Copyright &copy; Company.com</div>
    </div>
</body>
</html>
```

Fork me on GitHub

page1.html

```
<!DOCTYPE html>
<html data-layout-decorate="layout">
<body>
    <div data-layout-fragment ="content">
        <h1>This is the page 1</h1>
        <p>Lorem ipsum ...</p>
    </div>
</body>
</html>
```