FPT Education
FPT UNIVERSITY

Java

Spring Web Development

Objectives

- Building web applications using the Spring Framework.
  - Spring MVC (Model-View-Controller)
  - Spring WebFlux
  - Spring Boot
  - RESTful Web Services
  - Security
  - Integration with other technologies
- Understand a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text (Thymeleaf)

Spring Web Development Introduction

- The Spring Framework offers a robust and flexible platform for developing web applications, with comprehensive support for integration, scalability, security, and testing.
- With Spring Framework, a strong community support and alignment with modern architectural patterns make it a compelling choice for building modern and reliable web applications.

Spring Web Development Introduction

- Spring provides a comprehensive and flexible platform for building web applications, whether you're using traditional MVC patterns or adopting newer reactive programming paradigms.
  - Spring MVC (Model-View-Controller)
  - Spring WebFlux
  - Spring Boot
  - RESTful Web Services
  - Security
  - Integration with other technologies
  - Testing

Spring MVC (Model-View-Controller)

- Spring MVC is the traditional approach to building web applications with Spring.
- It provides a robust architecture for organizing your web application into components such as controllers, views, and models.
- Controllers handle incoming requests, models represent the data, and views render the response.

Spring WebFlux

- **Spring WebFlux** is the reactive approach introduced in Spring 5.
- It's based on Project Reactor and the Reactive Streams specification.
- WebFlux allows you to build non-blocking, reactive web applications, which are particularly useful for handling high concurrency and streaming data scenarios.
- The reactive-stack web framework, Spring WebFlux is fully non-blocking, supports Reactive Streams (https://www.reactive-streams.org/) back pressure, and runs on such servers as Netty, Undertow, and Servlet containers.

Spring Boot

- While not exclusively for web development, Spring Boot greatly simplifies the process of creating stand-alone, production-grade Spring-based applications.
- Consider using Spring Boot to quickly bootstrap your Spring applications.
- Spring Boot offers a convention-over-configuration approach, simplifying the setup and configuration of Spring-based applications.
- Spring Boot is an excellent choice for rapidly developing web applications with minimal setup and boilerplate code.

Spring Security

- Security: Spring Security is the de facto standard for securing Spring-based applications.
- It provides comprehensive security features for web applications, including authentication, authorization, session management, and more.
- It supports both traditional server-rendered applications (like Spring MVC) and modern single-page applications (like those built with Angular or React).

Spring REST

- If your web applications require RESTful services, consider using Spring's support for building REST APIs.
- Spring's REST capabilities, combined with Spring MVC or Spring WebFlux, provide powerful tools for creating and consuming RESTful services.
- Spring MVC and Spring WebFlux can both be used to create RESTful services, with features like content negotiation, request mapping, and input validation.

Integration with other technologies

- Spring integrates seamlessly with other technologies commonly used in web development, such as databases (via Spring Data), messaging systems (via Spring Messaging), and frontend frameworks (via RESTful APIs or WebSocket support).

Spring Data

- For data access, Spring Data offers a range of options, including JPA, MongoDB, Redis, and more.
- Choose the appropriate Spring Data module based on your data storage requirements to simplify data access and manipulation within your web applications.

Templating engines

- When it comes to templating engines, consider using Thymeleaf or Freemaker.
- Both provide powerful template processing capabilities, enabling the seamless integration of dynamic content into web pages.
- Thymeleaf, in particular, is a popular choice for Spring-based web applications due to its natural integration with Spring MVC.

FPT Education
FPT UNIVERSITY

Java

Thymeleaf - a Java-based server side template engine

What is Thymeleaf?

- Thymeleaf is a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text.
- The main goal of Thymeleaf is to provide an elegant and highly-maintainable way of creating templates.
- Thymeleaf improves communication of design and bridges the gap between design and development teams.
- Thymeleaf has also been designed from the beginning with Web Standards in mind - especially HTML5 - allowing you to create fully validating templates if
- that is a need for you

Kinds of templates in Thymeleaf

- There are two markup template modes (HTML and XML), three textual template modes (TEXT , JAVASCRIPT and CSS) and a no-op template mode (RAW).
- The **HTML template** mode will allow any kind of HTML input, including HTML5, HTML 4 and XHTML. No validation or well-formedness check will be performed, and template code/structure will be respected to the biggest possible extent in output.
- The **XML template** mode will allow XML input. In this case, code is expected to be well-formed – no unclosed tags, no unquoted attributes, etc – and the parser will throw exceptions if well-formedness violations are found. Note that no validation (against a DTD or XML Schema) will be performed.

Kinds of templates in Thymeleaf

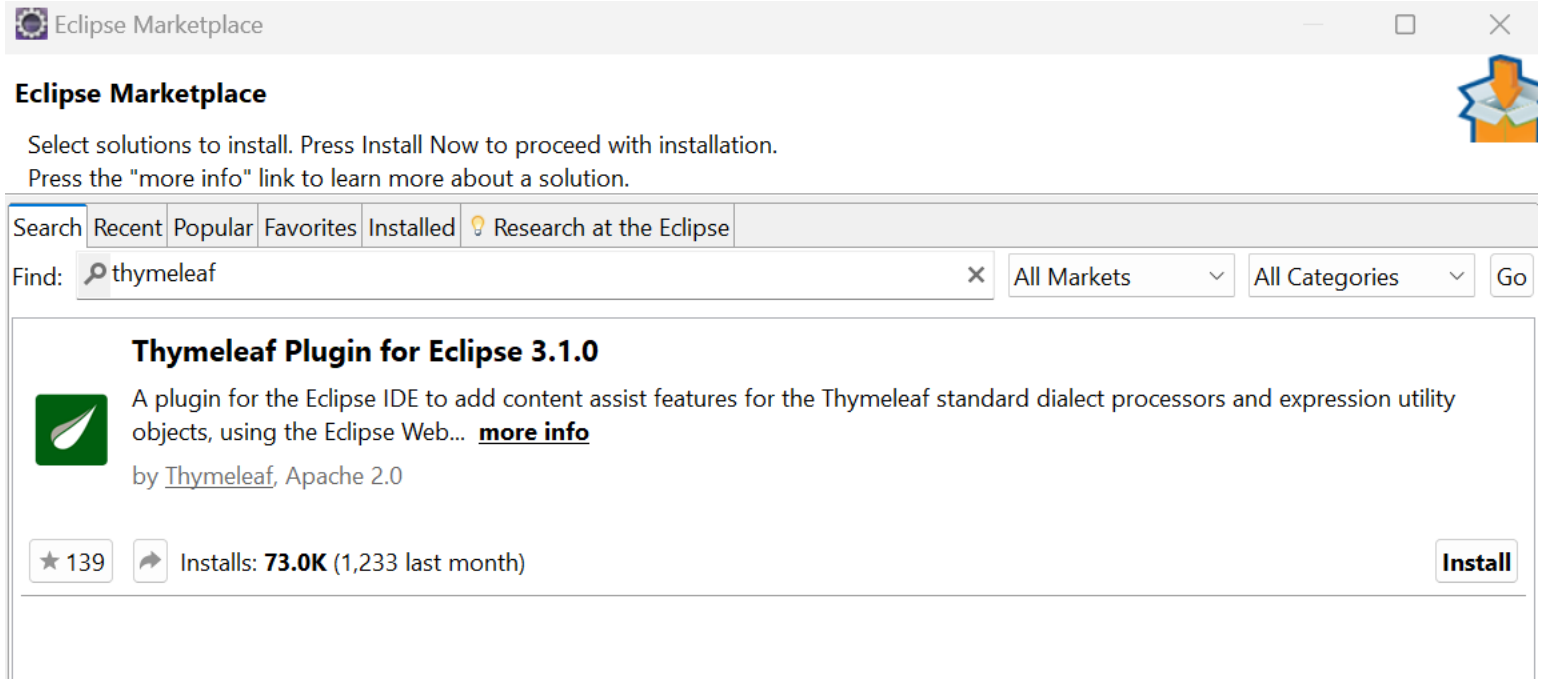
- The **TEXT template** mode will allow the use of a special syntax for templates of a non-markup nature. Examples of such templates might be text emails or templated documentation. Note that HTML or XML templates can be also processed as TEXT , in which case they will not be parsed as markup, and every tag, DOCTYPE, comment, etc, will be treated as mere text.
- The **JAVASCRIPT template** mode will allow the processing of JavaScript files in a Thymeleaf application. This means being able to use model data inside JavaScript files in the same way it can be done in HTML files, but with JavaScript-specific integrations such as specialized escaping or natural scripting. The JAVASCRIPT template mode is considered a textual mode and therefore uses the same special syntax as the TEXT template mode.

Kinds of templates in Thymeleaf

- The CSS template mode will allow the processing of CSS files involved in a Thymeleaf application. Similar to the JAVASCRIPT mode, the CSS template mode is also a textual mode and uses the special processing syntax from the TEXT template mode.

- The RAW template mode will simply not process templates at all. It is meant to be used for inserting untouched resources (files, URL responses, etc.) into the templates being processed. For example, external, uncontrolled resources in HTML format could be included into application templates, safely knowing that any Thymeleaf code that these resources might include will not be executed.

Install Thymeleaf Plugin for Eclipse



Standard Expression Syntax

- Simple expressions:
  - Variable Expressions: ${...}
  - Selection Variable Expressions: *{...}
  - Message Expressions: #{...}
  - Link URL Expressions: @{...}
  - Fragment Expressions: ~{...}
- Literals
  - Text literals: 'one text' , 'Another one!' ,…
  - Number literals: 0 , 34 , 3.0 , 12.3 ,…
  - Boolean literals: true , false
  - Null literal: null
  - Literal tokens: one , sometext , main ,…

Standard Expression Syntax

- Text operations:
  - String concatenation: +
  - Literal substitutions: |The name is ${name}|
- Arithmetic operations:
  - Binary operators: + , - , * , / , %
  - Minus sign (unary operator): -
- Boolean operations:
  - Binary operators: and , or
  - Boolean negation (unary operator): ! , not

Standard Expression Syntax

- Comparisons and equality:
  - Comparators: > , < , >= , <= (gt , lt , ge , le)
  - Equality operators: == , != (eq , ne)
- Conditional operators:
  - If-then: (if) ? (then)
  - If-then-else: (if) ? (then) : (else)
- Default: (value) ?: (defaultvalue)
- Special tokens:
  - No-Operation: _

HTML Template Example

sayhello.html

```
1  <!DOCTYPE html>
2
3  <html xmlns:th="http://www.thymeleaf.org" xmlns:hello="http://hello">
4
5    <head>
6      <title>Say Hello! - Extending Thymeleaf in 5 minutes</title>
7      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
8    </head>
9
10   <body>
11
12     <h1>To the world...</h1>
13
14     <p hello:sayto="World">Hi ya!</p>
15
16     <h1>To each planet...</h1>
17
18     <ul>
19       <li th:each="planet : ${planets}" hello:saytoplanet="${planet}">Hi Planet X!</li>
20     </ul>
21
22   </body>
23 </html>
```

Attribute Precedence

Order	Feature	Attributes
1	Fragment inclusion	`th:insert` `th:replace`
2	Fragment iteration	`th:each`

3	Conditional evaluation	`th:if`
		`th:unless`
		`th:switch`
		`th:case`

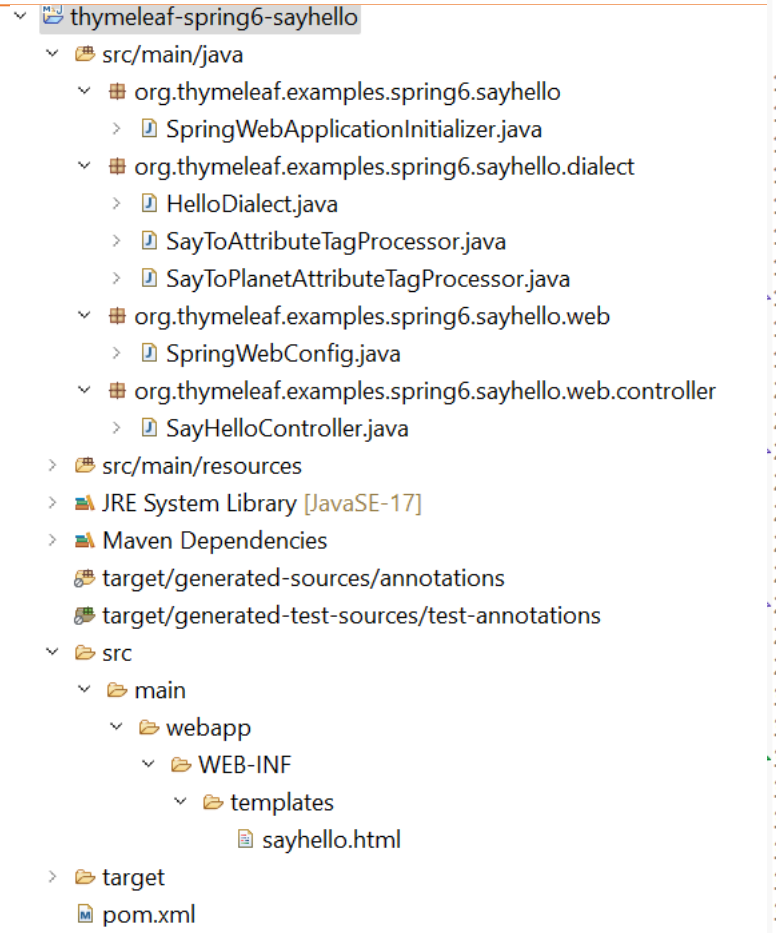
4	Local variable definition	`th:object` `th:with`

5	General attribute modification	`th:attr`
		`th:attrprepend`
		`th:attrappend`

6	Specific attribute modification	`th:value`
		`th:href`
		`th:src`
		`...`

7	Text (tag body modification)	`th:text` `th:utext`
8	Fragment specification	`th:fragment`
9	Fragment removal	`th:remove`

Create a simple project using Thymeleaf




```java
import jakarta.servlet.Filter;

public class SpringWebApplicationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    public static final String CHARACTER_ENCODING = "UTF-8";
    public SpringWebApplicationInitializer() {
        super();
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[] { SpringWebConfig.class };
    }

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class<?>[0];
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

    @Override
    protected Filter[] getServletFilters() {
        final CharacterEncodingFilter encodingFilter = new CharacterEncodingFilter();
        encodingFilter.setEncoding(CHARACTER_ENCODING);
        encodingFilter.setForceEncoding(true);
        return new Filter[] { encodingFilter };
    }
}
```

Controller

```java
import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;


@Controller
public class SayHelloController {

    public SayHelloController() {
        super();
    }


    @ModelAttribute("planets")
    public List<String> populatePlanets() {
        return Arrays.asList(new String[] {
                "Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"
        });
    }

    @RequestMapping({"/","/sayhello"})
    public String showSayHello() {
        return "sayhello";
    }
}
```

The Standard Dialect

- Thymeleaf is an extremely extensible template engine (in fact it could be called a template engine framework) that allows you to define and customize the way your templates will be processed to a fine level of detail.
- An object that applies some logic to a markup artifact (a tag, some text, a comment, or a mere placeholder if templates are not markup) is called a processor, and a set of these processors – plus perhaps some extra artifacts – is what a dialect is normally comprised of.
- Out of the box, Thymeleaf's core library provides a dialect called the Standard Dialect, which should be enough for most users.

The Standard Dialect

```java
import java.util.HashSet;

public class HelloDialect extends AbstractProcessorDialect {

    public HelloDialect() {
        super(
                "Hello Dialect",    // Dialect name
                "hello",            // Dialect prefix (hello:*)
                1000);              // Dialect precedence
    }


    /*
     * Initialize the dialect's processors.
     *
     * Note the dialect prefix is passed here because, although we set
     * "hello" to be the dialect's prefix at the constructor, that only
     * works as a default, and at engine configuration time the user
     * might have chosen a different prefix to be used.
     */
    public Set<IProcessor> getProcessors(final String dialectPrefix) {
        final Set<IProcessor> processors = new HashSet<IProcessor>();
        processors.add(new SayToAttributeTagProcessor(dialectPrefix));
        processors.add(new SayToPlanetAttributeTagProcessor(dialectPrefix));
        // This will remove the xmlns:hello attributes we might add for IDE \
        processors.add(new StandardXmlNsTagProcessor(TemplateMode.HTML, diale
        return processors;
    }
```

```java
import org.thymeleaf.context.ITemplateContext;

public class SayToAttributeTagProcessor extends AbstractAttributeTagProcessor {

    private static final String ATTR_NAME = "sayto";
    private static final int PRECEDENCE = 10000;

    public SayToAttributeTagProcessor(final String dialectPrefix) {
        super(
            TemplateMode.HTML, // This processor will apply only to HTML mode
            dialectPrefix,     // Prefix to be applied to name for matching
            null,              // No tag name: match any tag name
            false,             // No prefix to be applied to tag name
            ATTR_NAME,         // Name of the attribute that will be matched
            true,              // Apply dialect prefix to attribute name
            PRECEDENCE,        // Precedence (inside dialect's precedence)
            true);             // Remove the matched attribute afterwards
    }

    protected void doProcess(
            final ITemplateContext context, final IProcessableElementTag tag,
            final AttributeName attributeName, final String attributeValue,
            final IElementTagStructureHandler structureHandler) {

        structureHandler.setBody(
                "Hello, " + HtmlEscape.escapeHtml5(attributeValue) + "!", false);
    }
}
```

Attribute Tag Processor

```
4  import org.thymeleaf.IEngineConfiguration;
15
16 public class SayToPlanetAttributeTagProcessor extends AbstractAttri
17
18     private static final String ATTR_NAME = "saytoplanet";
19     private static final int PRECEDENCE = 10000;
20
21     private static final String SAYTO_PLANET_MESSAGE = "msg.hellopl
22
23     public SayToPlanetAttributeTagProcessor(final String dialectPre
34
35     protected void doProcess(
77 }
```

```
23     public SayToPlanetAttributeTagProcessor(final St
24         super(
25             TemplateMode.HTML, // This processor wil
26             dialectPrefix,     // Prefix to be appli
27             null,              // No tag name: match
28             false,             // No prefix to be ap
29             ATTR_NAME,         // Name of the attrib
30             true,              // Apply dialect pref
31             PRECEDENCE,        // Precedence (inside
32             true);             // Remove the matched
33     }
```

```
35     protected void doProcess(
36             final ITemplateContext context, final IProcessableElementTag tag,
37             final AttributeName attributeName, final String attributeValue,
38             final IElementTagStructureHandler structureHandler) {
39
40
41         final IEngineConfiguration configuration = context.getConfiguration();
42
43         final IStandardExpressionParser parser =
44                 StandardExpressions.getExpressionParser(configuration);
45
46         final IStandardExpression expression = parser.parseExpression(context, attributeValue);
47
48         final String planet = (String) expression.execute(context);
49
50
51         final String i18nMessage =
52                 context.getMessage(
53                         SayToPlanetAttributeTagProcessor.class,
54                         SAYTO_PLANET_MESSAGE,
55                         new Object[] {planet},
56                         true);
57
58         structureHandler.setBody(HtmlEscape.escapeHtml5(i18nMessage), false);
59     }
60 }
```

Spring Web Configuration


```java
import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.thymeleaf.examples.spring6.sayhello.dialect.HelloDialect;
import org.thymeleaf.spring6.SpringTemplateEngine;
import org.thymeleaf.spring6.templateresolver.SpringResourceTemplateResolver;
import org.thymeleaf.spring6.view.ThymeleafViewResolver;
import org.thymeleaf.templatemode.TemplateMode;

@Configuration
@EnableWebMvc
@ComponentScan
public class SpringWebConfig implements WebMvcConfigurer, ApplicationContextAware {
    private ApplicationContext applicationContext;

    public SpringWebConfig() {
        super();
    }

    public void setApplicationContext(final ApplicationContext applicationContext) throws BeansException {
        this.applicationContext = applicationContext;
    }
```

Spring Web Configuration

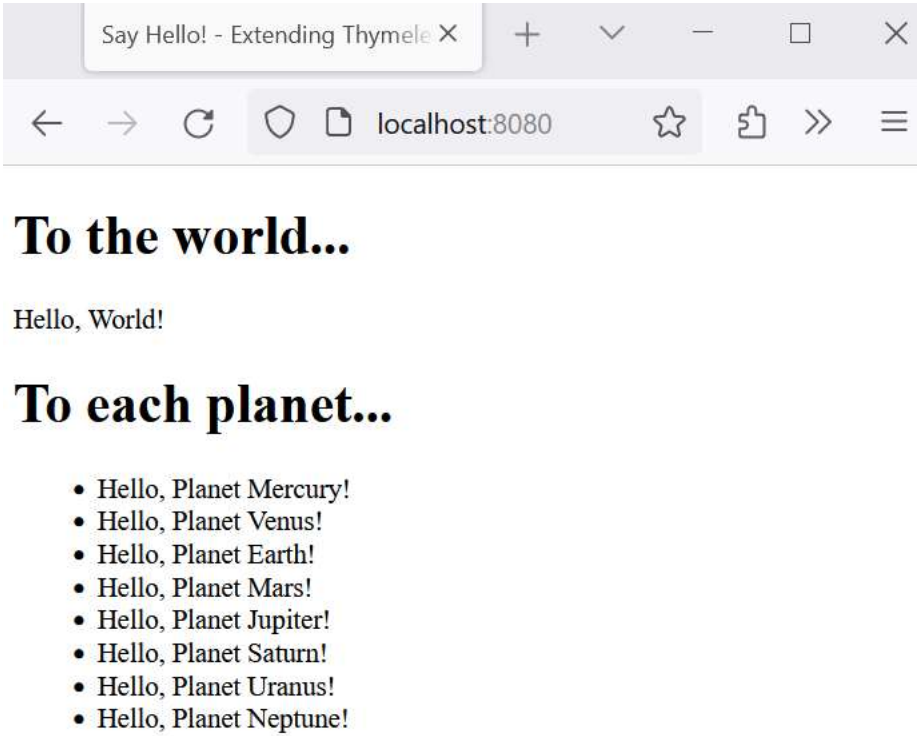
```
34    @Bean
35    public ResourceBundleMessageSource messageSource() {
36        ResourceBundleMessageSource resourceBundleMessageSource = new ResourceBundleMessageSource();
37        resourceBundleMessageSource.setBasename("Messages");
38        return resourceBundleMessageSource;
39    }
40
41    /* ********************************************************** */
42    /*  THYMELEAF-SPECIFIC ARTIFACTS                              */
43    /*  TemplateResolver <- TemplateEngine <- ViewResolver        */
44    /* ********************************************************** */
45
46    @Bean
47    public SpringResourceTemplateResolver templateResolver(){
48        SpringResourceTemplateResolver templateResolver = new SpringResourceTemplateResolver();
49        templateResolver.setApplicationContext(this.applicationContext);
50        templateResolver.setPrefix("/WEB-INF/templates/");
51        templateResolver.setSuffix(".html");
52        templateResolver.setTemplateMode(TemplateMode.HTML);
53        // Template cache is true by default. Set to false if you want
54        // templates to be automatically updated when modified.
55        templateResolver.setCacheable(true);
56        return templateResolver;
57    }
```

Spring Web Configuration

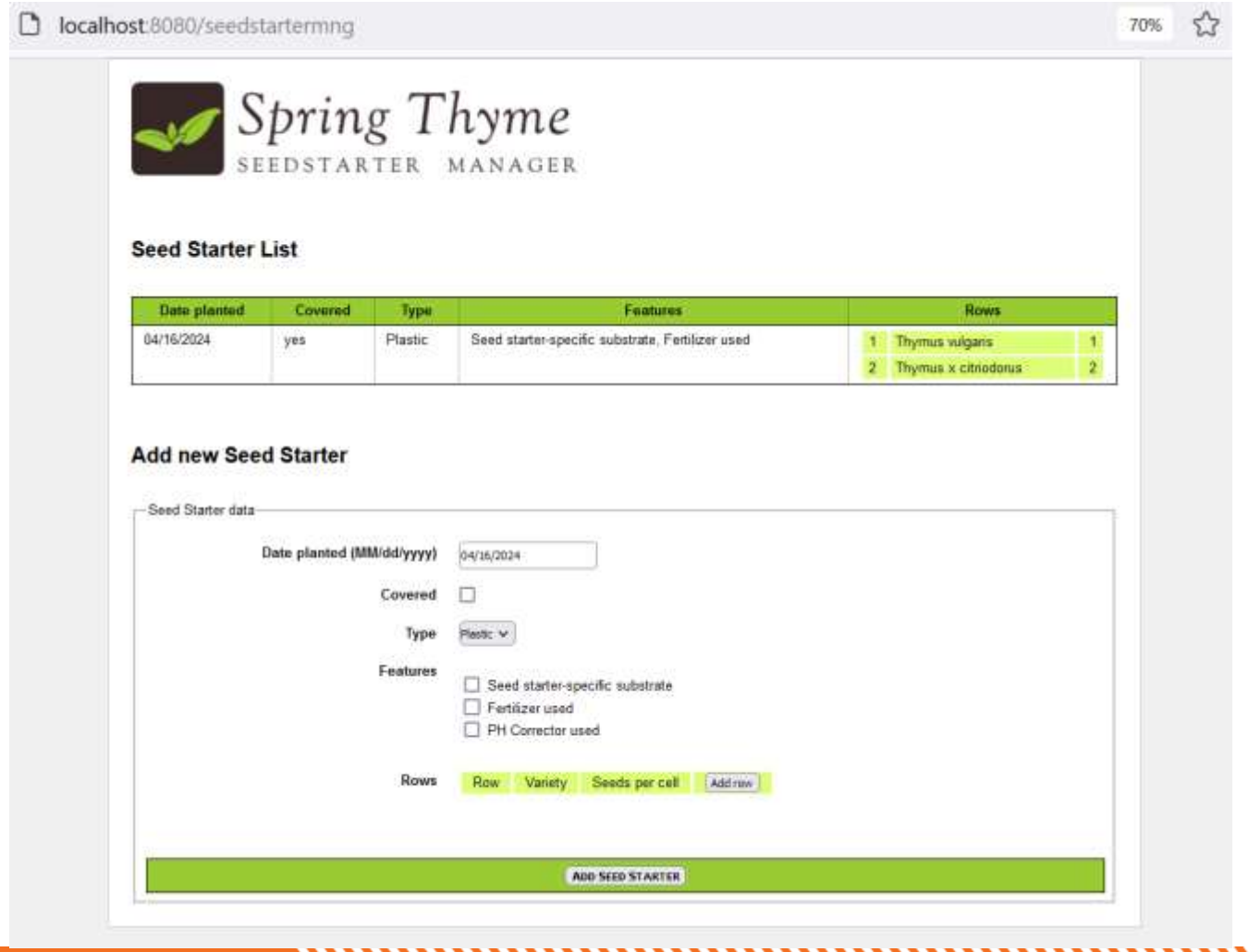

```
59      @Bean
60      public SpringTemplateEngine templateEngine(){
61          SpringTemplateEngine templateEngine = new SpringTemplateEngine();
62          templateEngine.setEnableSpringELCompiler(true); // Compiled SpringEL should speed
63          templateEngine.setTemplateResolver(templateResolver());
64          templateEngine.addDialect(new HelloDialect());
65          return templateEngine;
66      }
67
68      @Bean
69      public ThymeleafViewResolver viewResolver(){
70          ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
71          viewResolver.setTemplateEngine(templateEngine());
72          return viewResolver;
73      }
74
75      /* ************************************************************* */
76      /*  Defines callback methods to customize the Java-based configuration */
77      /*  for Spring MVC enabled via {@code @EnableWebMvc}             */
78      /* ************************************************************* */
79
80      /*
81       *  Dispatcher configuration for serving static resources
82       */
83      @Override
84      public void addResourceHandlers(final ResourceHandlerRegistry registry) {
85          WebMvcConfigurer.super.addResourceHandlers(registry);
86          registry.addResourceHandler("/images/**").addResourceLocations("/images/");
87          registry.addResourceHandler("/css/**").addResourceLocations("/css/");
88          registry.addResourceHandler("/js/**").addResourceLocations("/js/");
89      }
90
91  }
```

Run Project

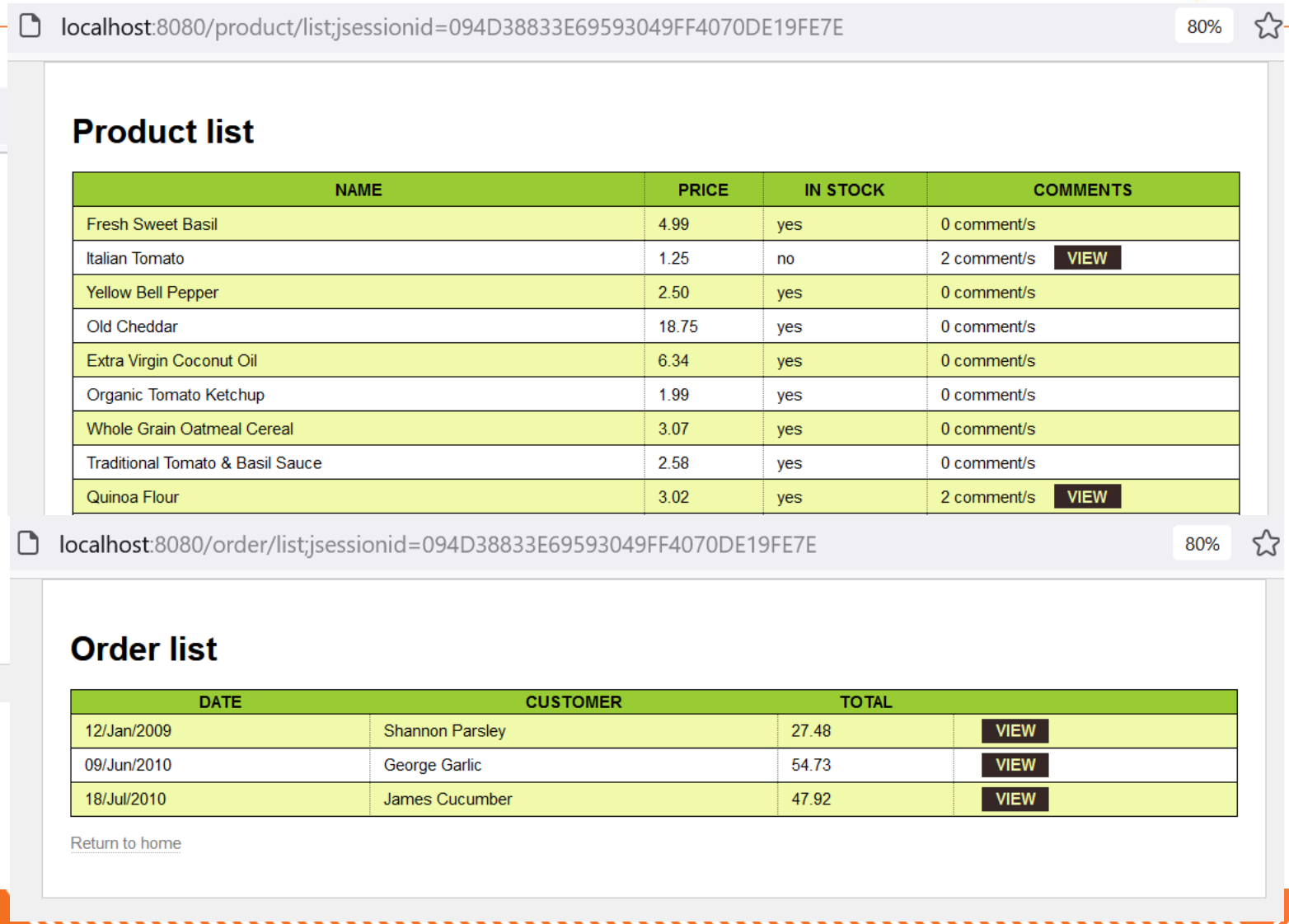
```
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:16 AM org.apache.catalina.core.ApplicationContext log
[INFO] [talledLocalContainer] INFO: Initializing Spring DispatcherServlet 'dispatcher'
[INFO] [talledLocalContainer] SLF4J: No SLF4J providers were found.
[INFO] [talledLocalContainer] SLF4J: Defaulting to no-operation (NOP) logger implementation
[INFO] [talledLocalContainer] SLF4J: See https://www.slf4j.org/codes.html#noProviders for further detai
[INFO] [talledLocalContainer] SLF4J: Class path contains SLF4J bindings targeting slf4j-api versions 1.
[INFO] [talledLocalContainer] SLF4J: Ignoring binding found at [jar:file:/D:/DemonstrationCode/thymelea
ples-spring6-sayhello/target/cargo/configurations/tomcat10x/webapps/ROOT/WEB-INF/lib/log4j-slf4j-impl-2
rBinder.class]
[INFO] [talledLocalContainer] SLF4J: See https://www.slf4j.org/codes.html#ignoredBindings for an explan
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:19 AM org.apache.catalina.startup.HostConfig deployWAR
[INFO] [talledLocalContainer] INFO: Deployment of web application archive [D:\DemonstrationCode\thymele
mples-spring6-sayhello\target\cargo\configurations\tomcat10x\webapps\ROOT.war] has finished in [8,470]
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:19 AM org.apache.catalina.startup.HostConfig deployDire
[INFO] [talledLocalContainer] INFO: Deploying web application directory [D:\DemonstrationCode\thymeleaf
les-spring6-sayhello\target\cargo\configurations\tomcat10x\webapps\host-manager]
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:19 AM org.apache.catalina.startup.HostConfig deployDire
[INFO] [talledLocalContainer] INFO: Deployment of web application directory [D:\DemonstrationCode\thyme
xamples-spring6-sayhello\target\cargo\configurations\tomcat10x\webapps\host-manager] has finished in [9
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:19 AM org.apache.catalina.startup.HostConfig deployDire
[INFO] [talledLocalContainer] INFO: Deploying web application directory [D:\DemonstrationCode\thymeleaf
les-spring6-sayhello\target\cargo\configurations\tomcat10x\webapps\manager]
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:19 AM org.apache.catalina.startup.HostConfig deployDire
[INFO] [talledLocalContainer] INFO: Deployment of web application directory [D:\DemonstrationCode\thyme
xamples-spring6-sayhello\target\cargo\configurations\tomcat10x\webapps\manager] has finished in [82] ms
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:19 AM org.apache.coyote.AbstractProtocol start
[INFO] [talledLocalContainer] INFO: Starting ProtocolHandler ["http-nio-8080"]
[INFO] [talledLocalContainer] Apr 16, 2024 1:40:19 AM org.apache.catalina.startup.Catalina start
[INFO] [talledLocalContainer] INFO: Server startup in [9455] milliseconds
[INFO] [talledLocalContainer] Tomcat 10.0.20 started on port [8080]
[INFO] Press Ctrl-C to stop the container...
```



Example: Spring Thyme Seedstarter Manager



Example: *The Good Thymes Virtual Grocery*



Summary

Concepts were introduced:

- Building web applications using the Spring Framework.
  - Spring MVC (Model-View-Controller)
  - Spring WebFlux
  - Spring Boot
  - RESTful Web Services
  - Security, Integration with other technologies
- Amodern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text (Thymeleaf)