Haaga-Helia

Back End Programming

Views & Model

Haaga-Helia

Spring Boot

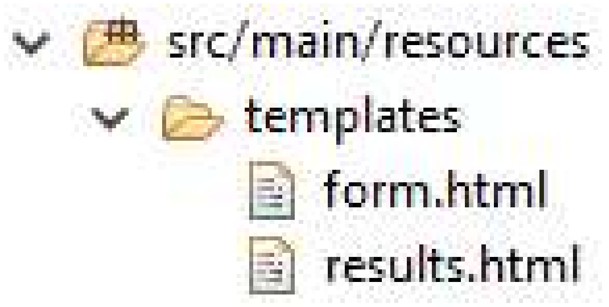
- Thymeleaf is used for the views during this course
- Thymeleaf is a modern server-side Java template engine for web and standalone environments
- www.thymeleaf.org
- How to start? Add dependency to pom.xml

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

Haaga-Helia

Spring Boot

- Thymeleaf templates are HTML files that also work as static prototypes
- With Spring Boot Thymeleaf templates are saved to `resources/templates` folder



```
src/main/resources
    templates
        form.html
        results.html
```

Spring Boot

- Thymeleaf template example

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Server Programming</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  </head>

  <body>
    <h1 th:text="'Hello ' + ${name}">Hello</h1>
  </body>
</html>
```

Spring Boot

- Accessing views
  - Controller handles request and returns the name of the View
  - Example below handels request for /index endpoint and returns view called ”index” (index.html Thymeleaf template)
  - Note! There is no @ResponseBody annotation when using Thymeleaf templates.

```
@Controller
public class MyController {
    @RequestMapping("/index")
    public String home() {
        // do something
        return "index";
    }
}
```

Haaga-Helia

Spring Boot

- The value of a parameter can be added to the ***Model*** object that makes it accessible to the view
- In a typical Spring application, Controller classes are responsible for preparing a model map with data and selecting a view to be rendered

```
…
import org.springframework.ui.Model;

@Controller
public class HelloController {
 @RequestMapping("/hello")
 public String greeting(@RequestParam(name="name") String name, Model model)   {
       model.addAttribute("name", name);
       return "hello";
  }
}
```

Haaga-Helia

Spring Boot

- In Thymeleaf, the model attributes can be accessed with the following syntax: **${attributeName}**
- Thymeleaf parses the template and evaluates *th:text* expression to render the value of the *${name}* parameter

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Server Programming with Spring Boot</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
    <p th:text="'Hello, ' + ${name} + '!'" />
</body>
</html>
```

Haaga-Helia

Spring Boot

- Model can contain the list of object which can be iterated and displayed as a table with Thymeleaf
- In the following example `messageRepository.findAll()` method returns the list of message objects

```
@RequestMapping("/message")
public String messages(Model model) {
  model.addAttribute("messages", messageRepository.findAll());
  return "messagelist";
}
```

Spring Boot

- Thymeleaf provides *th:each* attribute to iterate over the list of objects

```
<tr th:each="message : ${messages}">
  <td th:text="${message.id}">1</td>
  <td th:text="${message.msg}">Text ...</td>
</tr>
```

Haaga-Helia

Spring Boot

- GET request
  - Values are sent in URL in URL’s query string
- POST request
  - Values are sent in the request body
  - Typically used when sending a complete web form or uploading files
- How to define request type in controller?

```
@RequestMapping(value="/greeting", method=RequestMethod.POST)
```

OR

```
@RequestMapping(value="/greeting", method=RequestMethod.GET)
```

Spring Boot

- Instead of @RequestMapping annotation you can also use method specific shortcut annotations (`@GetMapping, @PostMapping` etc.)

```
@RequestMapping(value="/greeting", method=RequestMethod.POST)
```

EQUALS TO

```
@PostMapping("/greeting")
```

Spring Boot

- Following mapping allows the controller to differentiate the requests to the /hello (GET and POST requests)

```
@Controller
public class HelloController {
    @GetMapping("/hello")
    public String greetingForm(Model model) {
        model.addAttribute("message", new Message());
        return "hello";
    }

    @PostMapping("/hello")
    public String greetingSubmit(@ModelAttribute Message msg, Model model) {
        model.addAttribute("message", msg);
        return "result";
    }
}
```

Haaga-Helia

Spring Boot

- HTML Forms are needed when you want to collect data from the application end users
- A form will take input from the users and post it to a server

```
<form action="Script URL" method="GET|POST">
    form elements like input, dropdowns...
</form>
```

Haaga-Helia

Spring Boot: Form

- Thymeleaf form example

```
<form action="#" th:action="@{/hello}" th:object="${message}“
method="post">
    <p>Id: <input type="text" th:field="*{id}" /></p>
     <p>Message: <input type="text" th:field="*{msg}" /></p>
      <p><input type="submit" value="Submit" /></p>
</form>
```

- `th:action=`**"@{/hello}"** expression directs the form to POST to the /hello endpoint
- `th:object="${`**message}"** expression is the model object used to collect data. We need to create Message class next.

Spring Boot: Form

- Message class

```
public class Message {

    private long id;
    private String msg;

          ... getters an setters
}
```

Haaga-Helia

Spring Boot: Form

- Controller handles the form submit
- The msgSubmit() method is mapped to POST

```
...
@PostMapping("/hello")
public String msgSubmit(@ModelAttribute Message msg, Model model)
{
    model.addAttribute("message", msg);
    return "redirect:/result";
}
...
```

- It is recommended to use redirect afer POST. That prevents duplicate form submissions (PostRedirectGet = PRG)

Spring Boot: Form

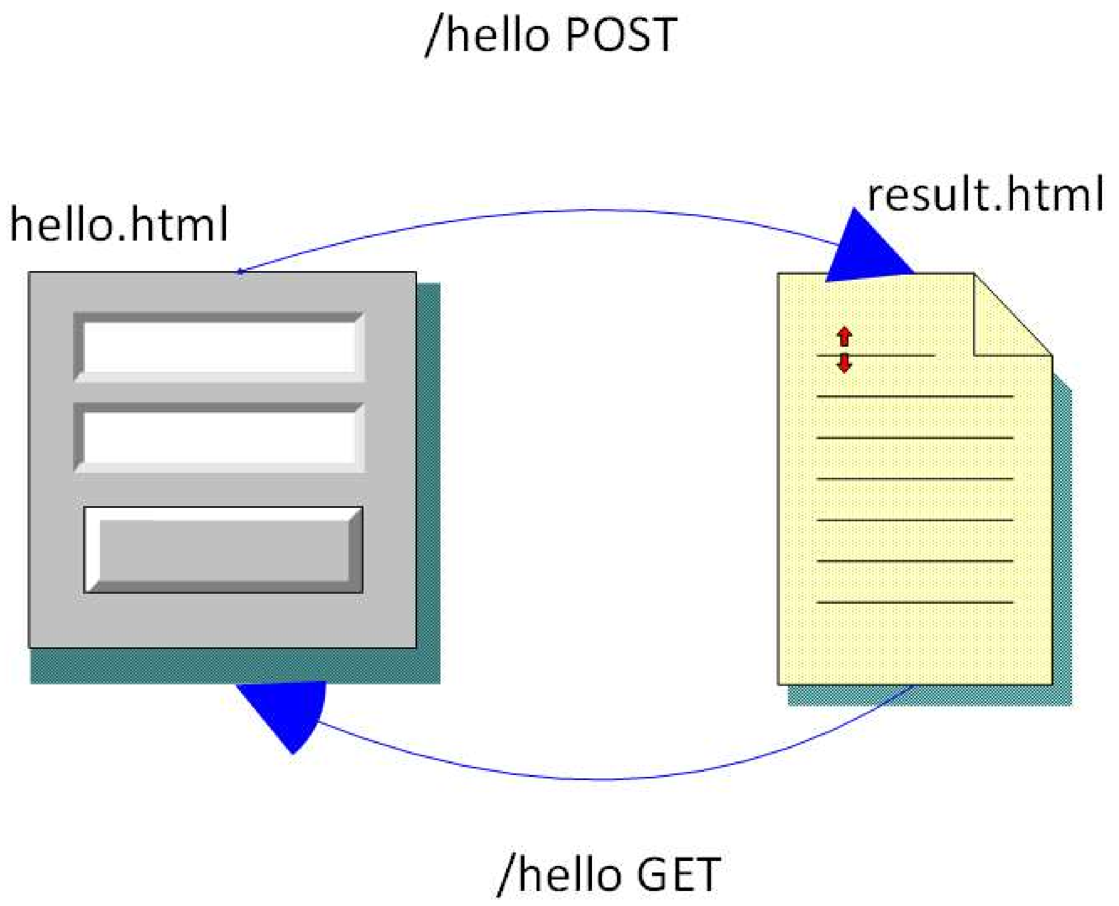
- Finally we need Thymeleaf template for showing results (result.html)

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Server Programming</title>
    <meta http-equiv="Content-Type" content="text/html; 
charset=UTF-8" />
</head>
<body>
    <p th:text="'id: ' + ${message.id}" />
    <p th:text="'content: ' + ${message.msg}" />
    <a href="/hello">Submit another message</a>
</body>
</html>
```

Haaga-Helia

Spring Boot: Form

- HelloForm example



Spring Boot: Form validation

- Validation: Class attributes can be flagged with standard validation attributes (=Bean validation)

```
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;

public class Message {
  @NotNull
  private long id;

  @Size(min=2, max=30)
  private String name;

  ...getters & setters
}
```

Spring Boot: Form validation

- POM.XML: insert validation depencency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

- Controller: Add new arguments to controller request methdod. BindingResult object is used to check validation result. @Valid attribute gather attributes filled out in the form.

```
@RequestMapping(value="/hello", method=RequestMethod.POST)
public String greetingSubmit(@Valid Message msg, BindingResult bindingResult
        , Model model) {
    if (bindingResult.hasErrors()) {
      return "hello";
    }
model.addAttribute("message", msg);
   return "result";
}
```

Spring Boot: Form validation

- Thymeleaf provides validation function `#fields.hasErrors()` which can be used to check if field contains any validation errors
- Example

```
<tr>
  <td>Message: <input type="text" th:field="*{name}" /></td>
  <td th:if="${#fields.hasErrors('name')}" th:errors="*{name}">Error</td>
</tr>
```

Spring Boot

- Demo codes
  1. HelloForm
     - Simple form example
  2. HelloFormValidation
     - Simple form example with validation

See 'How to run course demos' instruction from the course Moodle site.