infoShare ACADEMY

Invest in Pomerania ACADEMY

Aplikacja Webowa: Thymeleaf Część 2

Fundusze Europejskie Program Regionalny
Rzeczpospolita Polska
URZĄD MARSZAŁKOWSKI WOJEWÓDZTWA POMORSKIEGO
Unia Europejska Europejski Fundusz Rozwoju Regionalnego

HELLO

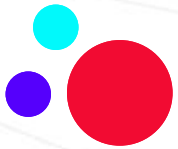
Paweł Dobrzański

Software developer z ponad 15-letnim stażem.

infoShare ACADEMY

Agenda

1. Podsumowanie składni
2. Pętle, warunki i wyrażenia
3. Formularze
4. Grajmy! (zdążymy?)



Ćwiczenie 0

- Przygotuj środowisko programistyczne (IntelliJ).
- Pobierz projekt z repozytorium:

```
git clone https://github.com/infoshareacademy/...
```

- Zweryfikuj czy uruchamia się poprawnie.

01. Podsumowanie składni

Wiedza w pigułce

1. Podsumowanie składni Thymeleaf

Składnia	Użycie	Przykład
@{}	Do przekazania adresu URL	`th:href="@{http://google.pl}"` `th:src="@{/img/picture.png}"`
${}	Podstawowe wywołanie zmiennej	`th:text="${expression}"` `th:text="${user.name}"`
*{}	Odniesienie do pola obiektu	`th:object="${user}"` `th:text="*{name}"`
~{}	Odwołanie do fragmentu	`th:fragment="footer"` `th:replace="~{fragments::footer}"`
#{}	Odwołanie do statycznych tekstów (przydatne przy wielojęzyczności)	`welcome.message=Hello World!` `th:text="#{welcome.message}"`

02. Pętle, warunki i wyrażenia

Wnieśmy programowanie do HTML-a ☺

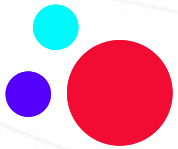
2. Pętle, warunki i wyrażenia

- If – Unless.

```
<td>
    <span th:if="${baby.gender == 'F'}">It's a girl!</span>
    <span th:unless="${baby.gender == 'F'}">Baby boy incoming.</span>
</td>
```

- Switch – Case.

```
<td th:switch="${queue.size()}">
    <span th:case="0">Queue is empty, you can send your message immediately.</span>
    <span th:case="1">Your message will be next in line.</span>
    <div th:case="*">
        We are sorry, you need to wait approximately <span th:text="${queue.size()}"></span> minutes.
    </div>
</td>
```



2. Pętle, warunki i wyrażenia

- Ternary / Elvis operator

```
<td th:class="${teacher.active} ? 'bg-success' : 'bg-danger'"
    th:text="${teacher.status} ?: 'UNKNOWN'"></td>
```

- Ternary działa tak samo jak w Javie (warunek musi zwracać boolean).
- Elvis występuje w bardziej zaawansowanych językach jak Groovy czy Kotlin.
- Elvis musi konwertować wartość na boolean zachowując oryginalną wartość.

- Niepełny ternary operator

```
<li th:class="${page == 'home'} ? 'font-weight-bold'"></li>
```

Ćwiczenie 5

- Korzystając z instrukcji warunkowych, zmodyfikuj fragment nawigacji swojej strony (**<nav>**) w taki sposób, aby podstrona, która jest aktualnie wyświetlana była wizualnie oznaczona.
- Np. jeśli używasz buttonów, skorzystaj z klas **btn-outline-*** jako domyślnych, natomiast **btn-*** dla aktywnej strony.
- Pamiętaj, że:
  - **th:class** nadpisuje wszelkie klasy css elementu.
  - **th:classappend** dodaje nowe klasy css do istniejących.

2. Pętle, warunki i wyrażenia

- Pętla **th:each**.
- Można również zastosować zmienną statusu.
- Zawiera kilka przydatnych wartości:
  - **index**: indeks iteracji (rozpoczyna się od 0)
  - **count**: numer iteracji (licząc od 1)
  - **size**: łączna liczba elementów listy
  - **even/odd**: sprawdza czy indeks jest parzysty
  - **first**: sprawdza czy aktualna iteracja jest pierwsza
  - **last**: sprawdza czy aktualna iteracja jest ostatnia

```
<tr th:each="student: ${students}">
    <td th:text="${student.id}"></td>
    <td th:text="${student.name}"></td>
</tr>
```

```
<div class="row">
    <div class="col-12 mb-3" th:each="opt, stat : ${options}"
         th:classappend="${stat.size % 2 == 0 or !stat.first} ? 'col-md-6'">
        <a th:href="${opt.url}" class="btn btn-lg w-100 py-5 text-nowrap"
           th:classappend="${stat.odd} ? 'btn-light' : 'btn-outline-dark'">
            <span th:text="${stat.count}"></span>. <span th:text="${opt.name}"></span>
        </a>
    </div>
</div>
```

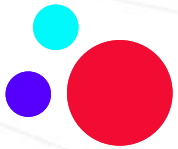
infoShare ACADEMY

2. Pętle, warunki i wyrażenia – th:each

```
model.addAttribute("words",
    List.of("Let", "the", "Force", "be", "with", "you"));
```

```
<tr th:each="word, stat : ${words}">
```

word	index	count	size	even	odd	first	last
Let	0	1	6	true	false	true	false
the	1	2	6	false	true	false	false
Force	2	3	6	true	false	false	false
be	3	4	6	false	true	false	false
with	4	5	6	true	false	false	false
you	5	6	6	false	true	false	true



2. Pętle, warunki i wyrażenia

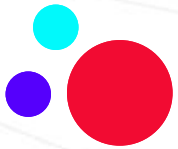
- Obsługa Enum w **th:each**.

```
public enum Color {
    BLACK, BLUE, RED, YELLOW, GREEN, ORANGE, PURPLE, WHITE
}
```

- W szablonie:

```
<select name="color">
    <option th:each="colorOpt : ${T(com.infoshareacademy.thymeleaf.model.Color).values()}"
            th:value="${colorOpt}" th:text="${colorOpt}"></option>
</select>
```

- Operator **T** służy do odwołania do klasy lub statycznych metod.
- Wada – musimy wpisać pełną ścieżkę pakietu klasy.



2. Pętle, warunki i wyrażenia

- Jeśli chcemy użyć Enum z opisem.
- W szablonie odwołujemy się po prostu do metody:

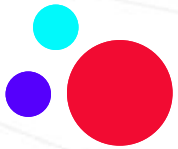
```
<select name="color">
    <option
        th:each="colorOpt : ${T(com.infoshareacademy.thymeleaf.model.Color).values()}"
        th:value="${colorOpt}" th:text="${colorOpt.displayValue}"></option>
</select>
```

```
public enum Color {
    BLACK( displayValue: "Black"),
    BLUE( displayValue: "Blue"),
    RED( displayValue: "Red"),
    YELLOW( displayValue: "Yellow"),
    GREEN( displayValue: "Green"),
    ORANGE( displayValue: "Orange"),
    PURPLE( displayValue: "Purple"),
    WHITE( displayValue: "White");

    private final String displayValue;

    Color(String displayValue) {
        this.displayValue = displayValue;
    }

    public String getDisplayValue() {
        return displayValue;
    }
}
```



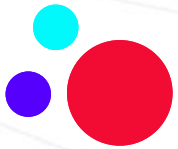
2. Pętle, warunki i wyrażenia

- Jak pozbyć się uciążliwości wpisywania pełnego pakietu?
- Z pomocą przyjdzie specjalne wyrażenie **__${}__**
- Wyrażenie tak zapisane zostanie ewaluowane pierwsze, a następnie wyliczy się wyrażenie zewnętrzne.
- W kontrolerze:

```
model.addAttribute("enum", Color.class.getName());
```

- W szablonie:

```
<select name="color">
    <option th:each="colorOpt : ${T(__${enum}__).values()}"
            th:value="${colorOpt}" th:text="${colorOpt.displayValue}"></option>
</select>
```



Ćwiczenie 6

- Do klasy Player dodaj nowe pole **level**, które będzie typu **PlayerLevel**.
- Utwórz Enum **PlayerLevel** z wartościami: Beginner, Advanced, Expert.
- Przebuduj **PlayerService** tak, aby przechowywał listę graczy.
- Niech przy inicjalizacji (konstruktor) wstawi się do niej 5 obiektów z kolejnymi ID i dowolnymi (niepustymi i unikatowymi) imionami oraz dowolnymi poziomami zaawansowania.
- Skoryguj widok **/players** i dodaj do tabelki dodatkową kolumnę „level”.

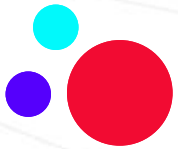
03. Formularze

Składam podanie

3. Formularze

- Formularz w thymeleaf niewiele się różni od czystego html.

```
<form th:action="@{/saveStudent}" th:object="${student}" method="post">
    <table>
        <tr>
            <td><label for="phone" th:text="#{msg.phone}"></label></td>
            <td><input type="number" th:field="*{phone}" /></td>
        </tr>
        <tr>
            <td><label for="name" th:text="#{msg.name}"></label></td>
            <td><input type="text" th:field="*{name}" /></td>
        </tr>
        <tr>
            <td><input type="submit" value="Submit" /></td>
        </tr>
    </table>
</form>
```



3. Formularze

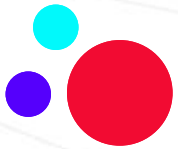
- Odbiór w kontrolerze odbywa się poprzez **@PostMapping** oraz adnotację **@ModelAttribute**.

```
@PostMapping("/saveStudent")
String saveStudent(@ModelAttribute Student form) {
    service.saveStudent(form);
    return "redirect:/students";
}
```

- Czasem potrzeby jest wrapper np. **StudentForm**.
- Dobrą praktyką jest wykonywać **redirect** po udanym zapisie.
- Warto pokusić się o walidację danych, weryfikację poprawności zapisu i w razie błędu można ponownie wyświetlić formularz z komunikatem błędu.

Ćwiczenie 7

- Utwórz nowy widok **/players/edit**, w którym wyświetlisz formularz dodawania nowego gracza (metoda kontrolera oraz szablon html).
- W formularzu dajemy jedynie dwa pola:
  - Tekstowe – imię
  - Lista rozwijana (select) – poziom zaawansowania (level)
- Na dole umieść dwa przyciski:
  - Anuluj – powróci do listy graczy
  - Zapisz – prześle formularz (Uwaga, zarówno **a**, **button** i **input** mogą korzystać ze styli **btn btn-***)
- Po wysłaniu formularza niech nastąpi sprawdzenie czy imię już nie występuje i czy nie jest puste (do walidacji utwórz metodę w **PlayerService**).
- ID powinno zostać nadane następne w kolejności (warto przebudować konstruktor statyczny w klasie **Player**).
- Jeśli wystąpią błędy niech ponownie zostanie wyświetlony formularz wraz z informacją o błędzie.
- Po poprawnym dodaniu gracza, niech nastąpi przekierowanie na listę graczy.



3. Formularze

- Edycja praktycznie nie różni się od dodawania.
- Potrzeba jedynie mieć odniesienie czy obsługujemy już istniejący obiekt.

```
<form th:action="@{/saveStudent}" th:object="${student}" method="post">
    <input type="hidden" th:field="*{id}">
    <table>
        <tr>
            <td><label for="phone" th:text="#{msg.phone}"></label></td>
            <td><input type="number" th:field="*{phone}" /></td>
        </tr>
        <tr>
            <td><label for="name" th:text="#{msg.name}"></label></td>
            <td><input type="text" th:field="*{name}" /></td>
        </tr>
        <tr>
            <td><input type="submit" value="Submit" /></td>
        </tr>
    </table>
</form>
```

3. Formularze

- Do samego wywołania formularza edycji możemy użyć sparametryzowanego URL oraz adnotacji **@PathVariable**.
- Możemy założyć, że wartość 0 oznacza czysty formularz i dodawanie nowego obiektu.

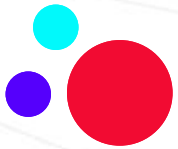
```
@GetMapping("/students/edit/{studentId}")
String editStudent(Model model, @PathVariable String studentId) {
    model.addAttribute("student", service.getStudent(Long.parseLong(studentId)));
    return "/student-edit";
}
```

Ćwiczenie 8

- W tabelce na widoku **/players** dodaj kolumnę z przyciskami kierującymi do edycji wybranego gracza.
- Uwaga: budowanie **href** ze zmienną jest nieco podchwytliwe: `th:href="${'/users/edit/'+user.id}"`
- Przebuduj metodę zapisu, aby w zależności od istnienia gracza o danym ID tworzyła nowego, bądź aktualizowała istniejącego.
- Pamiętaj o sprawdzeniu unikatowości imienia.

04. Grajmy!

Tylko gry jeszcze nie ma ☹



4. Grajmy

- Do stworzenia samej gry zabrakło już czasu, ale oto jakie kroki powinniśmy podjąć:
  - Utworzyć obiekt reprezentujący planszę i przechowujący jej stan.
  - Formularz rozpoczęcia gry – wybór graczy z listy.
  - Widok planszy i aktywnej gry.
  - Obsługa ruchów przez wysyłanie prostych requestów ze zmiennymi (numer pola / rodzaj ruchu).
  - Każdorazowe sprawdzenie czy gra się jeszcze toczy i ewentualne ogłoszenie zwycięzcy – modyfikacja statystyk zwycięzcy.

- Proponuję spróbować dokończyć aplikację implementując grę w kółko i krzyżyk ☺

DZIĘKI ZA UWAGĘ

Ciąg dalszy nastąpi...

Pytania kierujcie na Slack: @Pawel.Dobrzanski

infoShareAcademy.com

KONTAKT

kontakt@infoShareAcademy.com

(+48) 123 123 123

www.infoshareacademy.com

infoShare ACADEMY