infoShare ACADEMY

Invest in Pomerania ACADEMY

Aplikacja Webowa: Thymeleaf Część 1

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

HELLO

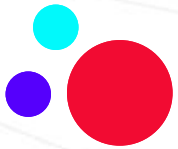
Paweł Dobrzański

Software developer z ponad 15-letnim stażem.

infoShare ACADEMY

Agenda

1. Instalacja.
2. Podstawowy szablon.
3. Przekazywanie zmiennych.
4. Podstrony i fragmenty.
5. Podsumowanie.



Ćwiczenie 0

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

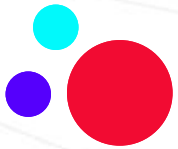
```
git clone https://github.com/infoshareacademy/...
```

- Zweryfikuj czy uruchamia się poprawnie.

infoShare ACADEMY

Thymeleaf – instalacja

Jak zacząć?



1. Instalacja

- Z pomocą przychodzi **spring initializr** - https://start.spring.io/
- Wygodny wybór opcji / wersji.
- Wyszukiwarka zależności.
- Automatycznie generuje szkielet projektu.
- Pobieramy gotowy do uruchomienia projekt – out-of-the-box.

infoShare ACADEMY

spring initializr

Project

- ○ Gradle - Groovy
- ○ Gradle - Kotlin
- ● Maven

Language

- ● Java
- ○ Kotlin
- ○ Groovy

Spring Boot

- ○ 3.1.0 (SNAPSHOT)
- ○ 3.1.0 (RC1)
- ○ 3.1.0 (M2)
- ○ 3.0.7 (SNAPSHOT)
- ● 3.0.6
- ○ 2.7.12 (SNAPSHOT)
- ○ 2.7.11

Project Metadata

Group	com.infoshareacademy
Artifact	thymeleaf
Name	thymeleaf
Description	JJDZR11 Thymeleaf
Package name	com.infoshareacademy.thymeleaf
Packaging	● Jar ○ War
Java	○ 20 ● 17 ○ 11 ○ 8

Dependencies — ADD DEPENDENCIES... CTRL + B

Thymeleaf TEMPLATE ENGINES

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

1. Instalacja

- Dodanie do istniejącego projektu.

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

infoShare ACADEMY

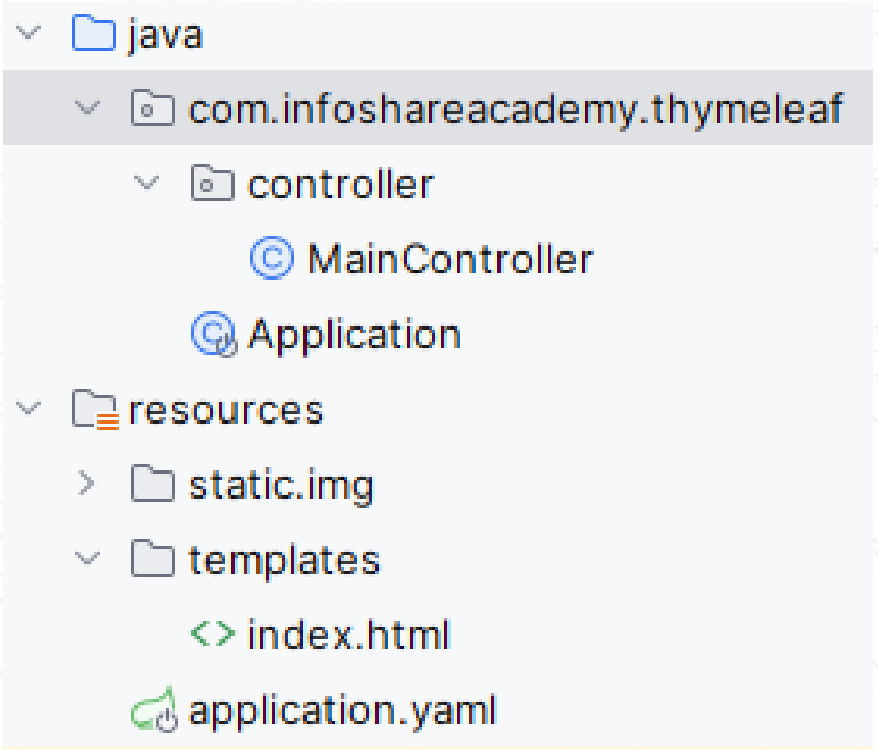
02. Podstawowy szablon

Webowe Hello World

2. Podstawowy szablon

Do wyświetlenia strony potrzebujemy minimum 2 elementów:

- Kontrolera.
- Szablonu widoku.



2. Podstawowy szablon - kontroler

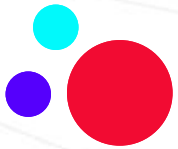
- Kontroler jest „pierwszą linią wsparcia” po wysłaniu żądania http.
- Definiujemy w nim tzw. **routing** – mapowanie URL na metodę obsługującą żądanie.
- Metody mapujące są typu **String** i mają zwracać nazwę szablonu dla Thymeleaf.

```
@Controller
public class MainController {

    @GetMapping("/")
    String index() {
        return "index";
    }

}
```

infoShare ACADEMY



2. Podstawowy szablon

- Szablon to klasyczny plik HTML.
- Może zawierać dodatkowe atrybuty z przedrostkiem **th:**
- Są one oznaczeniem dla Thymeleaf.
- Adresy oznaczamy składnią **@{}**.

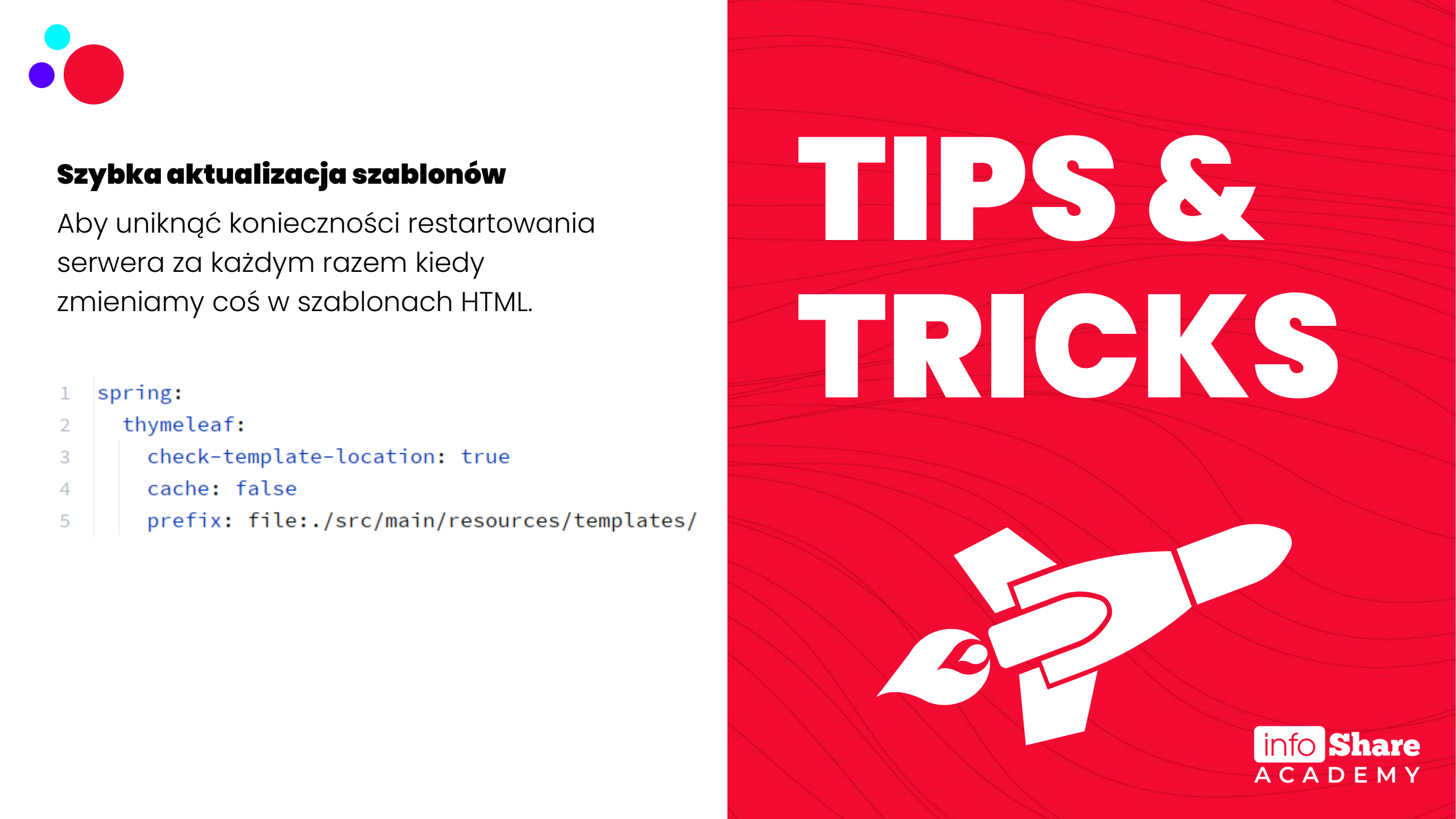
```
1  <!DOCTYPE html>
2  <html lang="pl">
3  <head>
4      <meta charset="UTF-8">
5      <link rel="icon" th:href="@{/img/spring-48px.png}" sizes="48x48" />
6      <title>JJDZR11 Aplikacja Webowa z Thymeleaf</title>
7  </head>
8  <body>
9      <h1>Witaj świecie!</h1>
10 </body>
11 </html>
```

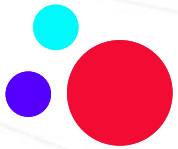
infoShare ACADEMY

Szybka aktualizacja szablonów

Aby uniknąć konieczności restartowania serwera za każdym razem kiedy zmieniamy coś w szablonach HTML.

```yaml
spring:
  thymeleaf:
    check-template-location: true
    cache: false
    prefix: file:./src/main/resources/templates/
```





Ćwiczenie 1

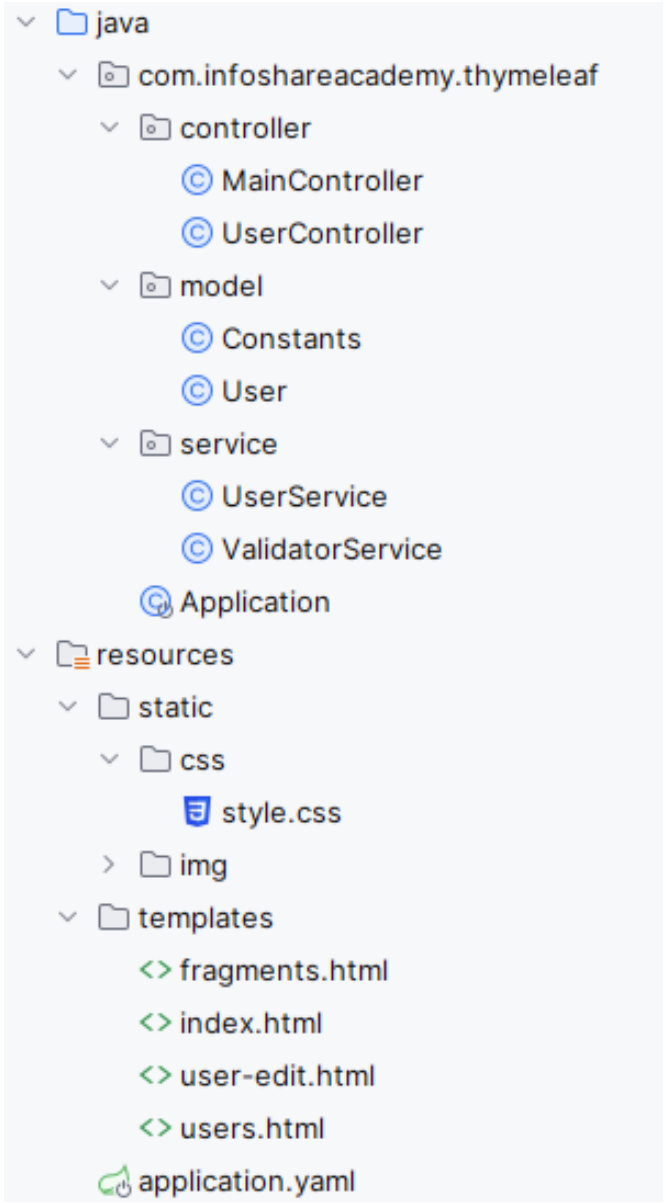
- Utwórz drugi szablon **info.html** poprzez skopiowanie index.html.
- Zmodyfikuj go, aby wyświetlał tytuł „Informacje o stronie” oraz jakiś tekst zamiast powitania <h1> (np. kilka słów o autorze).
- W kontrolerze dodaj metodę, która ścieżkę **/info** będzie kierować na szablon **info**.
- W obu szablonach HTML dodaj linki (**<a href=…>**), które będą odpowiednio przenosić nas na drugą stronę.

* **Bootstrap** mile widziany ☺

infoShare ACADEMY

Struktura projektu

Lepiej trzymać porządek, niż sprzątać.

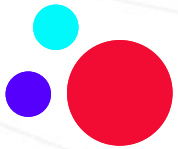


TIPS & TRICKS

InfoShare ACADEMY

Przekazywanie zmiennych

Podstawa Thymeleaf

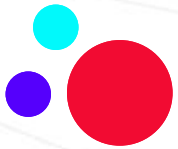


3. Przekazywanie zmiennych

- Do komunikacji między warstwami kontroler-widok służy klasa **Model**.
- Wystarczy wstawić ją w parametr metody mapującej.
- Przypisujemy zmienną za pomocą **addAttribute()**.
- Można łańcuchowo ☺

```
@GetMapping("/")
String index(Model model) {
    model.addAttribute( attributeName: "currentTime", System.currentTimeMillis());
    String expression = "Ala ma kota";
    model.addAttribute( attributeName: "expression", expression);
    return "index";
}
```

infoShare ACADEMY

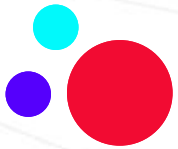


3. Przekazywanie zmiennych

- Do wstawienia wartości do szablonu używamy **th:text**.
- Odwołujemy się do nazwy atrybutu poprzez **${}**.
- Możemy przekazywać nie tylko proste wartości, ale i złożone obiekty.

```
<body>
    <h1>Witaj świecie!</h1>
    <h2>Czas systemowy: <span th:text="${currentTime}"></span></h2>
    <p th:text="${expression}"></p>
</body>
```

infoShare ACADEMY

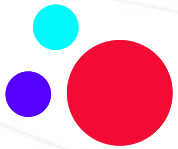


3. Przekazywanie zmiennych

- Możemy przygotować wszystkie atrybuty w formie **Mapy**.
- Przekażemy je za jednym razem.
- Przypisujemy zmienną za pomocą **addAllAttributes()**.
- Rozwiązanie ma wadę – podpowiadanie przez IntelliJ ☹

```
@GetMapping("/")
String index(Model model) {
    model.addAllAttributes(Map.of(
        k1: "currentTime", System.currentTimeMillis(),
        k2: "expression", v2: "Ala ma kota"
    ));
    return "index";
}
```

infoShare ACADEMY



Ćwiczenie 2

- W widoku **index** zmień treść powitania na „Witaj gościu! Mamy właśnie: xxx”, gdzie **xxx** wyświetli aktualną datę i godzinę (format dowolny).
- Oczywiście przekaż tę wartość w kontrolerze.
- Dla widoku **info** wykonaj refactor treści:
  - Utwórz pakiet **model**, a w nim klasę **Constants**.
  - Zadeklaruj w niej stałą (public static final String) zawierającą treść informacji na stronę **info**.
  - W kontrolerze **info** przekaż zawartość stałej.
  - W szablonie **info** odpowiednio wstaw przekazaną wartość.

infoShare ACADEMY

3. Przekazywanie zmiennych c.d.

- Przekazany obiekt pozwala na dostęp do swoich własności.
- Thymeleaf dostarcza uproszczony dostęp do pól obiektu (**.name** zamiast **.getName()**).
- Jeśli przekazujemy tylko jeden obiekt danej klasy, możemy pominąć nazwę atrybutu.

```
@GetMapping("/profile")
String profile(Model model) {
    User profile = userService.getProfile();
    model.addAttribute(profile);
    return "profile";
}
```

```
<h2>Profil użytkownika</h2>
<table class="table table-striped">
    <tr>
        <td>ID</td>
        <td th:text="${user.id}"></td>
    </tr>
    <tr>
        <td>Imię</td>
        <td th:text="${user.name}"></td>
    </tr>
    <tr>
        <td>Wiek</td>
        <td th:text="${user.age}"></td>
    </tr>
</table>
```

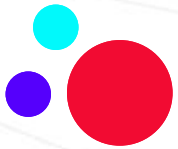
infoShare ACADEMY

3. Przekazywanie zmiennych

- Aby nie powtarzać wielokrotnie nazwy obiektu, możemy przypisać go w całości do danego elementu HTML.
- Służy do tego **th:object**.
- Pozwala to odnosić się wewnątrz do samych nazw pól poprzez składnię ***{}**.

```
<h2>Profil użytkownika</h2>
<table class="table table-striped" th:object="${user}">
    <tr>
        <td>ID</td>
        <td th:text="*{id}"></td>
    </tr>
    <tr>
        <td>Imię</td>
        <td th:text="*{name}"></td>
    </tr>
    <tr>
        <td>Wiek</td>
        <td th:text="*{age}"></td>
    </tr>
</table>
```

infoShare ACADEMY



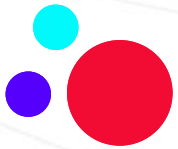
Ćwiczenie 3

- W pakiecie **model** utwórz klasę **Player**, wraz z polami:
  - long id
  - String name
  - int gamesPlayed
  - int gamesWon
- Utwórz pakiet **service**, a w nim klasę **PlayerService** z adnotacją **@Service**.
- W **PlayerService** utwórz dwa pola do przechowywania obiektów graczy i utwórz obiekty z podanymi na start ID oraz imionami.
- W **PlayerService** utwórz metodę getPlayer(long id) zwracającą obiekt gracza z danym ID.
- W kontrolerze wstrzyknij obiekt serwisu poprzez adnotację **@Autowire**.
- Utwórz nowy widok **/players** – metodę kontrolera oraz template.
- Niech widok prezentuje wszystkie dane obu graczy w formie tabeli.
- Dodaj link do tego widoku na stronie głównej (**index**).

info Share ACADEMY

04. Podstrony i fragmenty

Stosownie do zasady DRY



4. Podstrony i fragmenty

- Zawsze chcemy unikać powtarzania elementów. Czy to kod czy szablon html.
- Witryny internetowe zazwyczaj budowane są na jednym, wspólnym szablonie.
- Z pomocą przychodzą **th:fragment** oraz **th:replace**:

```
<head th:fragment="head">
    <meta charset="UTF-8">
    <link rel="icon" th:href="@{/img/spring-48px.png}" sizes="48x48" />
    <title>JJDZR11 Aplikacja Webowa z Thymeleaf</title>
</head>
```

```
<!DOCTYPE html>
<html lang="pl">
<head th:replace="~{fragments::head}"></head>
<body>
```

infoShare ACADEMY

4. Podstrony i fragmenty

Widoki rozproszone

fragments.html

- th:fragment=head
- th:fragment=navbar
- th:fragment=footer

index.html

- doctype
- html
- head th:replace
- body
- navbar th:replace
- --- **content** ---
- footer th:replace
- /body
- /html

page1.html

- doctype
- html
- head th:replace
- body
- navbar th:replace
- --- **content** ---
- footer th:replace
- /body
- /html

page2.html

- doctype
- html
- head th:replace
- body
- navbar th:replace
- --- **content** ---
- footer th:replace
- /body
- /html

Dwa podejścia

Jeden główny szablon

fragments.html

- th:fragment=head
- th:fragment=navbar
- th:fragment=footer

main.html

- doctype
- html
- head th:replace
- body
- navbar th:replace
- div th:replace=content
- footer th:replace
- /body
- /html

page1.html

- div th:fragment=content
- --- **content** ---
- /div

page2.html

- div th:fragment=content
- --- **content** ---
- /div

4. Podstrony i fragmenty

Widoki rozproszone

Czytelniejsze wywołanie szablonów

```
@GetMapping("/page1")
String page1() {
    return "page1";
}

@GetMapping("/page2")
String page2() {
    return "page2";
}
```

Dwa podejścia

Jeden główny szablon

Szablon main zawiera:

```
<div th:replace="~{${content}::content}"></div>
```

Natomiast metody kontrolerów:

```
@GetMapping("/page1")
String page1(Model model) {
    model.addAttribute("content", "page1");
    return "main";
}

@GetMapping("/page2")
String page2(Model model) {
    model.addAttribute("content", "page2");
    return "main";
}
```

ACADEMY

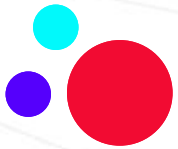
Ćwiczenie 4

- Wykorzystując wiedzę o **th:fragment** wykonaj refactor aplikacji tak, aby nie powtarzać tych samych elementów w różnych miejscach.
- Dodaj element nawigacji (**‹nav›**), który wyświetli się na każdej podstronie i pozwoli przełączać się pomiędzy innymi podstronami.
- Dodaj element stopki (**‹footer›**), który również wyświetli się na każdej podstronie, na samej dole strony i będzie zawierał jakieś minimalistyczne informacje (np. „Created by XXX, 2023”).
- Stopka powinna być widocznie oddzielona od treści i mieć pomniejszoną czcionkę (**Bootstrap** pomoże ☺).

infoShare ACADEMY

05. Podsumowanie

Wiedza w pigułce



5. Podsumowanie składni Thymeleaf

Składnia	Użycie	Przykład
@{}	Do przekazania adresu URL	`th:href="@{mailto:info@somesite.com}"` `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}"`

infoShare ACADEMY

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