UNIWERSYTET IM. ADAMA MICKIEWICZA W POZNANIU
Wydział Matematyki i Informatyki

Interfejs użytkownika w Spring Web

Czego potrzebujemy

Analogicznie do usług sieciowych potrzebujemy Spring Web.

Oprócz tego potrzebujemy jakiegoś frameworku do generowania stron w HTML. W Javie jest ich zatrzęsienie, do najpopularniejszych należą:

- Java Server Pages (odradzam, nie warto się męczyć)
- Java Server Faces (nie polecam z wielu powodów)
- FreeMarker (popularny, ale...)
- Apache Velocity (to jeszcze żyje...? podobno, tak)
- Vaadin (to właściwie coś więcej, niektórzy go kochają)
- Thymeleaf (polecam, tego użyjemy)

Czego potrzebujemy

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>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency...>

    <dependency...>
    <dependency...>
    <dependency...>
    <dependency...>
    <dependency...>
</dependencies>
```

Wzorzec projektowy MVC

Spring Web dostarcza możliwości budowania aplikacji zgodnie z wzorcem Model-View-Controller (MVC):

- kontroler (ang. Controller) pozwala zarządzać przepływem danych i przejściami między stronami
- widok (ang. View) to nic innego jak zrenderowana strona internetowa; w naszym wypadku wykorzystamy do tego celu szablony Thymeleaf
- model [danych] – to co przekazujemy do szablonu, aby wygenerować widok: Thymeleaf oferuje tu dużą dowolność

Dla dociekliwych

Abstrahując od możliwości budowy tzw. aplikacji jednostronicowych (ang. Single-page application), w AngularJS, React lub Vue.js, które możemy bezproblemowo zbudować przy użyciu MVC + REST, istnieją także inne wzorce, których można użyć (wszystko bowiem zależy od tego, co budujemy):

- Model-View-Presenter (MVP)
- Model-View-ViewModel (MVVM)

Model

Model (danych) jest realizowany przez klasę Model z pakietu org.springframework.ui, która przechowuje atrybuty:

- dowolnego typu
- jako parę klucz wartość
- generuje nazwę klucza (nazwa klasy atrybutu małymi literami), jeżeli “zapomnimy” dostarczyć własnej

Rzecz jasna możemy się do nich dostać wołając:

- getAttibute(*klucz*)
- containsAttribute(*klucz*)

Thymeleaf wykorzystuje to udostępniając atrybuty w szablonie.

View (Widok)

Widok realizujemy przez plik HTML z dodatkowymi, specyficznymi dla Thymeleaf atrybutami:

```
<body>
<p><a th:href="@{/}">Strona główna</a></p>
<div th:if="${#lists.isEmpty(ancestors)}">
    <h1 class="text-center"><strong>Nie ma przodków :(</strong></h1>
    <img style="align: center;"...>
</div>
<div th:if="!${#lists.isEmpty(ancestors)}">
    <h2 style="...">Przodkowie:</h2>
    <table class="table table-hover table-sm table-responsive">
        <thead class="thead-inverse"...>
        <tbody>
        <tr th:each="ancestor : ${ancestors}" th:class="${ancestor.sex.toLowerCase()}" th:classappend="ancestor"
            th:id="${ancestor.personId}">
            <td class="displayname" th:text="${ancestor.displayName}"></td>
            <td class="cntr birth" th:text="${ancestor.birth?.eventDay ?: 'NN'}"></td>
            <td class="cntr birth" th:text="${ancestor.birth?.eventMonth ?: 'NN'}"></td>
            <td class="cntr birth" th:text="${ancestor.birth?.eventYear ?: 'NN'}"></td>
            <td class="cntr marriage" th:text="${ancestor.marriage?.eventDay ?: 'NN'}"></td>
            <td class="cntr marriage" th:text="${ancestor.marriage?.eventMonth ?: 'NN'}"></td>
            <td class="cntr marriage" th:text="${ancestor.marriage?.eventYear ?: 'NN'}"></td>
            <td class="cntr death" th:text="${ancestor.death?.eventDay ?: 'NN'}"></td>
            <td class="cntr death" th:text="${ancestor.death?.eventMonth ?: 'NN'}"></td>
            <td class="cntr death" th:text="${ancestor.death?.eventYear ?: 'NN'}"></td>
            <td class="cntr" th:text="${ancestor.level}"></td>
        </tr>
        </tbody>
    </table>
</div>
<script th:inline="javascript">
    let descendant = [[${descendant}]];
```

View

Ciekawe atrybuty:

- th:href – pozwala wygenerować URI zależne od kontekstu
- th:if – wyrażenie warunkowe (w przykładzie skorzystałem także z wbudowanego atrybutu #lists)
- th:each – pozwala renderować listę wartości (tu: ancestors jest atrybutem typu List<AncestorData>, przekazanym w modelu)
- th:class – pozwala podać nazwę klasy (stosujemy wtedy, gdy ta ma zależeć od danych), generuje atrybut HTML *class*
- th:classappend – pozwala dołączyć dalsze klasy
- th:id – pozwala wygenerować atrybut HTML o nazwie *id* (potrzebne np. gdy chcemy modyfikować konkretne dane)
- th:text – pozwala wyświetlić żądany tekst

View

Ciekawe atrybuty (ciąg dalszy):

- th:inline – zaznacza że w wygenerowanym bloku trzeba wpisać przekazane atrybuty bezpośrednio w linię (`[[${descendant}]]` wpisuje po prostu atrybut o nazwie descendant podany w modelu (tu: po prostu id użytkownika z bazy danych))

Jest ich więcej, zachęcam do obejrzenia dokumentacji (lub przeszukania StackOverflow).

Zwracam uwagę na zastosowanie tzw. Elvis operator, tj. `?:`

Operator ten jest skrótem, który pozwala na przypisanie domyślnych wartości danym o wartości `null`.

Uwaga: Thymeleaf może renderować także inne typy szablonów, np. XML, JavaScript, CSS.

Controller (kontroler)

Analogicznie jak w przypadku usług sieciowych REST, możemy stworzyć klasę, która pod podanymi przez nas ścieżkami będzie zwracała „coś” (w tym wypadku widok):

- tym razem użyjemy adnotacji `@Controller`
- nie będziemy zwracali danych (nie bezpośrednio), dlatego typem zwracanym z metody będzie `String`.
- w takim kontrolerze da się zrobić obsługę usług sieciowych
  - `@RestController` to specyficznie skonfigurowany `@Controller`
  - nie należy jednak tego mieszać

Controller – przykład

```
@Controller
public class RootController {

    @NonNull GedcomImporter importer;
    @NonNull DbService dbService;

    @Autowired
    public RootController(@NonNull GedcomImporter importer, @NonNull DbService dbService) {...}

    @RequestMapping(path = "/", method = RequestMethod.GET)
    public String getHomePage(Model model) { return "index"; }

    @GetMapping(path = "/ancestors")
    public String getAncestorsForPerson(@RequestParam Long id,
                                        @RequestParam(defaultValue = "true") Boolean events,
                                        @RequestParam(defaultValue = "false") Boolean filter,
                                        Model model) {
        if (events) {
            List<AncestorsData> data = filter ?
                    dbService.findAncestorsData(id).stream()
                            .filter(AncestorsData::hasAtLeastOneFullEvent)
                            .toList() : dbService.findAncestorsData(id);
            model.addAttribute( attributeName: "ancestors", data);
            model.addAttribute( attributeName: "descendant", id);
            return "ancestors";
        }
        return "redirect:/";
    }
```

Controller (kontroler)

Warto zwrócić uwagę na:

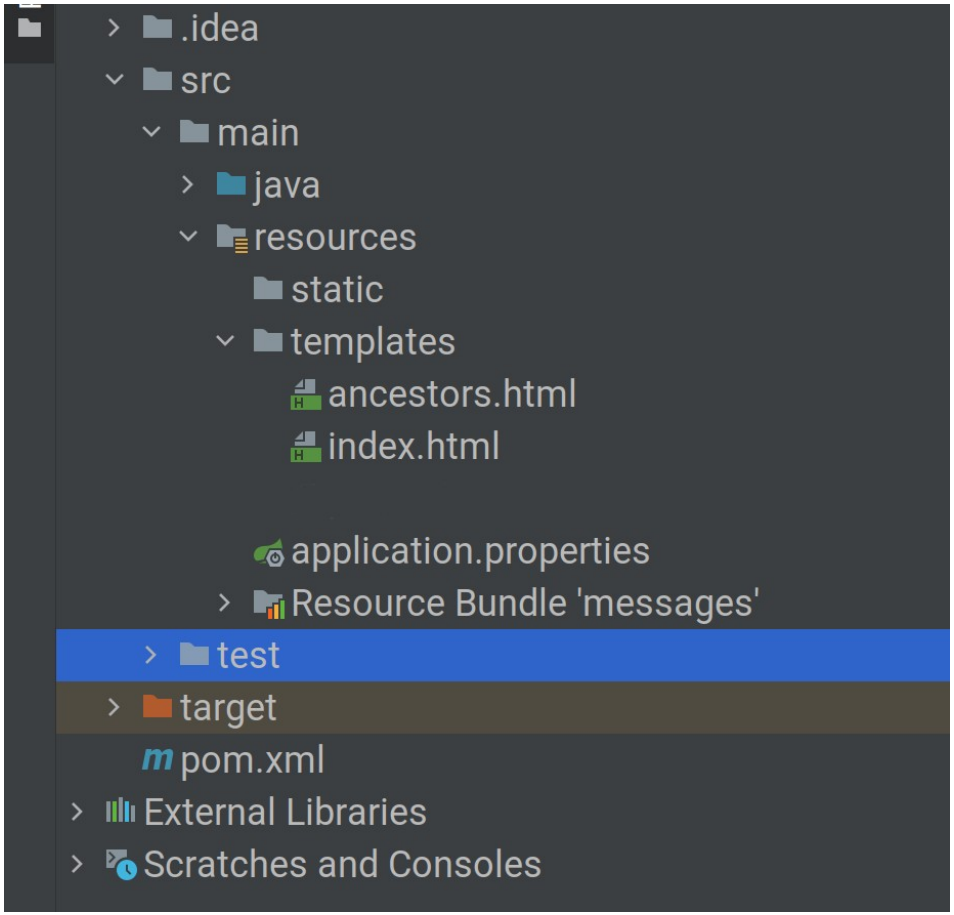
- sposób w jaki wskazujemy szablon widoku do zrenderowania: po prostu zwracamy jego nazwę bez rozszerzenia
- sposób obsługi przekierowań: `"redirect:ścieżka"`
- sposób w jaki tworzymy domyślne parametry w URL
- jak łączymy model i widok:
  - po prostu definiujemy parametr metody typu Model
  - Spring automatycznie wstrzykuje odpowiednią zależność
  - ustawiamy atrybuty w modelu (odpowiadają tym, używanym w szablonie)
  - Spring automatycznie konfiguruje Thymeleaf przekazując mu model

Struktura aplikacji internetowej

Pamiętają Państwo strukturę projektu Maven?

- src/main/java, src/main/resources, ...
- teraz main/resources się na coś przyda:
  - (innego niż przechowywanie pliku konfiguracyjnego aplikacji)
  - w podkatalogu static przechowujemy pliki statyczne
    - favicon.ico
    - CSS
    - pliki JavaScript
    - obrazki
    - (wszystkie są “serwowane” automatycznie)
  - w podkatalogu templates przechowujemy szablony widoku

Struktura aplikacji internetowej



Literatura Spring Web MVC

Literatura:

Serving Web Content with Spring MVC

Spring Boot CRUD Application with Thymeleaf | Baeldung

Accessing Data with JPA Tutorial

React.js and Spring Data REST

To (prawie) wszystko na dziś.