

Turistguide

Del 2 (Web Frontend)

1. Introduktion

I denne opgave skal I lave en web frontend til Turistguide projektet ved anvendelse af HTML og Thymeleaf. Det betyder, at **ResponseEntity** objekterne i forrige udgave af projektet skal udskiftes med visning af data på HTML sider.

2. Læringsmål

- ✓ At udarbejde dynamiske HTML sider med Thymeleaf
- ✓ Kan style HTML sider med CSS
- ✓ Kan Web Layer Slice test af Controlleren

3. Spring Boot projekt

Opret vha. Spring Initializr et **nyt** Spring Boot projekt, der hedder: **TouristGuide**.

- ✓ Inkluder Spring Web og Templates → Thymeleaf
- ✓ Husk at vælge Maven
- ✓ Pakkestrukturen skal være som forrige projekt

Kopier løbende genbrugbar kode fra **TouristGuideAPI** over i det nye projekt.

4. Overblik over endpoints

Krav til og forklaring på endpoints beskrives på de kommende sider.

GET	/attractions	GET attractions	▼
GET	/attractions/{name}	GET attractions/{name}	▼
GET	/attractions/{name}/tags	GET attractions/{name}/tags	▼
GET	/attractions/add	GET attractions/add	▼
POST	/attractions/save	POST attractions/save	▼
GET	/attractions/{name}/edit	GET attractions/{name}/edit	▼
POST	/attractions/update	POST attractions/update	▼
POST	/attractions/delete/{name}	POST attractions/delete/{name}	▼

5. Visning af alle turistattraktioner på en HTML side

Trin 1: Ændring af endpoint

Modificér endpointet `/attractions` fra den forrige version af projektet, så HTML siden `attractionList.html` vises med data fra endpointet (dvs. data fra repository klassens hardkodede ArrayList).

Bemærk: Dynamiske HTML sider placeres i templates mappen (under resources).

```
resources
├── static
│   └── main.css
└── templates
    └── attractionList.html
```

Trin 2: HTML side

Når siden vises, kan det f.eks. se således ud med data i en HTML tabel:

Turistattraktioner:	
Navn	Beskrivelse
SMK	Statens Museum for Kunst
Odense Zoo	Europas bedste zoo
Dyrehaven	Naturpark med skovområder
Tivoli	Forlystelsespark midt i København centrum

Trin 3: CSS Layout

Lav passende CSS styles for HTML siden (se bilag 3 for inspiration) og læg disse i filen `main.css` som placeres i static mappen under resources.

Et eksternt stylesheet inkluderes i Thymeleaf ved at tilføje flg. til HTML dokumentets head-sektion:

```
<link th:href="@{/main.css}" rel="stylesheet">
```

6. Visning af tags for turistattraktion på ny HTML side

Trin 1: Navigation til ny HTML side

Foretag ændring af [attractionList.html](#), så der vises en "Tags" knap ud for hver række. Når der trykkes på "Tags" navigeres til ny HTML side [tags.html](#).

Tilføj en knap i bunden af siden [attractionList.html](#) med teksten "Tilføj attraktion". I første omgang skal der ikke ske noget, når man trykker på knappen.

Skærbilledet kan f.eks. se sådan ud:

Turistattraktioner:		
Navn	Beskrivelse	
SMK	Statens Museum for Kunst	Tags
Odense Zoo	Europas bedste zoo	Tags
Dyrehaven	Naturpark med skovområder	Tags
Tivoli	Forlystelsespark midt i København centrum	Tags
Tilføj attraktion		

Trin 2: Nyt endpoint og ny HTML side

Tilføj endpointet: `/{{name}}/tags`, så den ny HTML side [tags.html](#) vises med tag data for den valgte turistattraktion (og en Tilbage knap, hvor man kommer tilbage til oversigten).

Tivoli tags	
Underholdning	
Restaurant	
Koncert	
Tilbage	

Trin 3: Ændring af Java klasser

TouristAttraction klassen tilføjes to nye attributter:

En liste til tags og en String til at registrere by (du skal bruge by i næste delopgave).

Hvert **TouristAttraction** objekt i repositoryklassen tilføjes en by og nogle tags¹.

F.eks.:

```
private List<TouristAttraction> attractions =  
    new ArrayList<> (List.of(  
        new TouristAttraction("SMK", "Museum for Kunst", "København", List.of("Kunst", "Museum")),  
        new TouristAttraction("Odense Zoo", "Europas bedste zoo", "Odense", List.of("Børnevenlig")),  
        new TouristAttraction("Dyrehaven", "Naturpark med skovområder", "Kongens Lyngby", List.of("Natur", "Gratis")),  
        new TouristAttraction("Tivoli", "Forlystelsespark i København centrum", "København", List.of("Børnevenlig"))  
    ));
```

7. Oprettelse af turistattraktion på ny HTML side

Der skal benyttes 2 endpoints til oprettelse af en ny turistattraktion:

- Først et nyt GET endpoint: **/add**, hvor der navigeres til en HTML side med en form, hvor data indtastes.
- Dernæst et POST endpoint: **/save**, som gemmer data i repository.

Formen kan f.eks. se sådan ud:

Tilføj turistattraktion

Navn:

Beskrivelse:

By:

Tags:

☐ Børnevenlig ☐ Gratis ☐ Kunst ☐ Museum ☐ Natur

¹ Du kan overveje at definere Tags i en enum.

Select- og Checkboxe skal udfyldes dynamisk, dvs. data hentes fra "databasen", når formen loades. Til dette formål skal der skrives to metoder i repositoryet:

```
public List<String> getCities() {
    // hent cities fra hardkodet liste i repository
}

public List<String> getTags() {
    // hent tags hardkodet liste i repository
}
```

Inspiration til initialisering af select- og checkboxe, samt udfyldning af model objekter med form data kan findes i bilag 1, 2.

8. Ændre af turistattraktion

Trin 1: Nye endpoints

Der skal 2 endpoints til redigering af en turistattraktion:

- GET endpoint: `/name/edit`, hvor der navigeres til en HTML side med den valgte turistattraktion, hvor data kan redigeres.
- POST endpoint: `/update`, som gemmer ændringerne i repository.

Trin 2: HTML sider

Tilføj ny "Opdater" knap på [attractionList.html](#), og vis også byen for hver attraktion (du kan lige så godt tilføje en "Slet" knap med det samme, selvom den først kommer i brug senere):

Turistattraktioner:

Navn	Beskrivelse	By			
SMK	Museum for Kunst	København	Opdater	Slet	Tags
Odense Zoo	Europas bedste zoo	Odense	Opdater	Slet	Tags

Tilføj attraktion

Opret en ny HTML side `updateAttraction.html`, hvor man kan redigere oplysningerne for en turistattraktion.

Det skal ikke være muligt at ændre navn².

Ændring af informationer

Navn:

Beskrivelse:

By:

Tags:
☐ Børnevenlig ☐ Gratis ☒ Kunst ☒ Museum ☐ Natur

9. Slet turistattraktion

Lav endpointet `/ {name} /delete` der sletter den valgte en turistattraktion i repository.

10. Lav en web layer slice test af controlleren

I skal teste jeres controller ved hjælp af Mockito og MockMVC.

11. EKSTRA-OPGAVER for de hurtige

- a) Arbejde med HTML og CSS forbedringer, såsom brug af favicon, navigationsmenu i toppen af alle websider, fragment og grid layout.

I kan finde eksempelkode til inspiration her (læs README.md for mere forklaring):
<https://github.com/Tine-m/extraFeatures-html-css>

- b) Ny funktionalitet:

Det skal være muligt at registrere billetpris for de turistattraktioner, hvor der skal betales entré.

² Se hvordan man laver tekstfelt readonly: https://www.w3schools.com/tags/att_input_readonly.asp

Det skal desuden være muligt at få vist prisen:

- i danske kroner, og
- valgfrit i euro (f.eks. ved tryk på kursberegner-knap).

Eksempelkode der henter valutakurser (læs README.md for mere forklaring):

<https://github.com/Tine-m/currencyRates>

Bilag 1: Inspirations kode til at udfylde og binde selectbox værdier til model objekt

[<select> - w3schools](#)

```
public class MyModel {
    private String selectedValue;
    // getters and setters
}

<form th:object="${myModel}" method="post">
    <select th:field="*{selectedValue}">
        <option value="">--Please select--</option>
        <option th:each="value : ${allValues}" th:value="${value}" th:text="${value}"></option>
    </select>
    <button type="submit">Submit</button>
</form>

@GetMapping("/")
public String myForm(Model model) {
    MyModel myModel = new MyModel();
    model.addAttribute("myModel", myModel);
    model.addAttribute("allValues", Arrays.asList("Value 1", "Value 2", "Value 3"));
    return "my-form";
}

@PostMapping("/")
public String myFormSubmit(@ModelAttribute MyModel myModel) {
    String selectedValue = myModel.getSelectedValue();
    // do something with the selected value
    return "redirect:/";
}
```

Bilag 2: Inspirations kode til at udfylde og binde checkbox værdier til model objekt

[input type checkbox - w3schools](#)

```
public class MyModel {
    private List<String> selectedValues;
    // getters and setters
}

<form th:object="${myModel}" method="post">
    <div th:each="value : ${allValues}">
        <input type="checkbox" th:field="*{selectedValues}" th:value="${value}" th:text="${value}" />
    </div>
    <button type="submit">Submit</button>
</form>

@GetMapping("/")
public String myForm(Model model) {
    List<String> startValues = Arrays.asList("Value 1", "Value 3");
    MyModel myModel = new MyModel();
    myModel.setSelectedValues(startValues);
    model.addAttribute("myModel", myModel);
    model.addAttribute("allValues", Arrays.asList("Value 1", "Value 2", "Value 3"));
    return "my-form";
}

@PostMapping("/")
public String myFormSubmit(@ModelAttribute MyModel myModel) {
    List<String> selectedValues = myModel.getSelectedValues();
    // do something with the selected values
    return "redirect:/";
}
```

Bilag 3: Inspiration til CSS styles

```
body {
  max-width: 500px;
  margin: auto;
  background: white;
  padding: 10px;
}

h1 {
  font-family: Arial;
  font-size: 1.5em;
}

label {
  width: 150px;
  display: inline-block;
}

table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
  padding: 5px;
}

th {
  text-align: left;
}

select, input[type="text"], input[type="number"]{
  width: 200px;
  box-sizing: border-box;
}
```