Exercises for IDATA2306 Application Development: Week #11

Alternatives: if you use language and/or framework other than Java Spring Boot, find the proper way to do corresponding things there!

Exercise 11.1 – Hello, Monolith app with ThymeLeaf [1h, fundamental]

The point of this exercise is to try out creating a monolith application with Spring Boot.

The task – migrate a static multi-HTML-file website to a Spring Boot project, where HTML code for navigation is stored only in one HTML file and included in all the pages where it is necessary.

Here is HTML+CSS starter-code you can use for copying content: https://github.com/strazdinsg/app-dev/tree/main/exercises/ex11.1-hello-monolith

It contains only a static site (frontend), no real data yet.

Steps:

1. Create a Spring Boot project.
2. Import dependency: `spring-boot-starter-thymeleaf`.
3. Import `spring-boot-devtools` dependency. This is needed to skip restarting the project every time you edit HTML templates.
4. Add the following lines in `application.properties` to configure ThymeLeaf templates and disable template caching (for easier development):
```
spring.thymeleaf.cache=false
spring.thymeleaf.prefix=file:src/main/resources/templates/
spring.thymeleaf.mode=HTML
spring.thymeleaf.encoding=UTF-8
spring.web.resources.cache.period=0
spring.web.resources.static-locations=file:src/main/resources/static/
```
5. Copy the HTML files (`index.hml`, `about.html`, `books.html`) from the provided starter-code to `src/main/resources/templates` in your project.
6. Copy the CSS file to `src/main/resources/static` folder.
7. Copy the images (the img folder) to `src/main/resources/static` folder.
8. Create a controller which will serve those pages:
    - You can name it `PageController`
    - Mark it with `@Controller` annotation (Note: `@Controller`, not `@RestController`)

- Create a method for each of the pages. Name them `getHome()`, `getBooks()`, `getAbout()`.
- Each of the methods must return a string – the name of the HTML template file with the content. For example, getHome should look like this:

```java
/**
 * The `Home` page.
 *
 * @return Name of the ThymeLeaf template to render
 */
@GetMapping("/")
public String getHome() {
  return "index";
}
```

This code means that whenever an `HTTP GET /` request is received, the content of `index.html` file will be returned.

9. Run the application. If you have done everything correctly, you should be able to access the home page in the browser (http://localhost:8080)
10. The navigation will not work, because the index.html file contains links to about.html and books.html, that's not correct. Let's fix it.
11. Edit the HTML files (`about.html`, `books.html`), change the URLs for anchor elements to `/about` and `/books` respectively.
12. Restart the app. Does the navigation work now?
13. If you look inside the HTML files, you will see that a lot of it is copy-paste code. Do some refactoring. As a minimum, extract the navigation to a separate file:
    a. Copy the content for the navigation (the whole <nav> element and all its descendants) to a separate file, for example: `src/main/resources/templates/components/navigation.html`
    b. In all the places where you want to include this navigation, place the following in the code (it effectively says "create a nav element here and the content for it must be taken from the file navigation.html"):
    ```html
    <nav th:insert="components/navigation"></nav>
    ```
14. Restart the application. Open the page in the browser and check if it still works.
15. Do you see any other sections which are the same in index.html, about.html and books.html? Any more things you could extract?

Exercise 11.2 – Show data in a monolith app [1.5h, Fundamental]

In this task you must insert real data in a static skeleton.

Your starter-code consists of two projects:

- Static HTML + CSS files provided here: https://github.com/strazdinsg/app-dev/tree/main/example-07-monolith/00-static
- An application with a database containing books, authors and genres, with necessary repositories and service classes: https://github.com/strazdinsg/app-dev/tree/main/exercises/ex11.2-monolith-with-data

Your end-result (the result of the exercise, when you are done), should be something like this: https://github.com/strazdinsg/app-dev/tree/main/example-07-monolith/01-monolith

Your task is to "walk the journey" from a static web to a complete Spring Boot app, with a database, which generates HTML pages based on the data. Remember: there is no point of cheating here, because the result is already provided. As Steve Jobs said: "The journey is the reward!"

Approximate steps/hints:

- Explore the given template files. These are the same files showed in the class, cleaned a bit.
- Explore the Spring boot starter files – the repositories, services.
- Incorporate the HTML and CSS files in you project. You can follow the steps in the previous exercises. At the end of this step, you should be able to start your Spring Boot app and see static pages in the browser.
- Add a `Model` argument to the controller methods. For example, the `getHome` method will look as follows now:

```
/**
  * The `Home` page.
  *
  * @param model The model where the data will be stored
  * @return Name of the ThymeLeaf template to render
  */
 @GetMapping("/")
 public String getHome(Model model) {
   return "index";
 }
```

- Add references to necessary service classes inside the controller (Hint: `BookService` and `AuthorService`)
- Select the necessary data by calling the relevant service methods. Add that data to the model (all the data within model will be accessible within the corresponding ThymeLeaf template). For example, the `getBooks` method may look something like this:

```
/**
 * The `Books` page.
 *
 * @param model The model where the data will be stored
 * @return Name of the ThymeLeaf template to render
 */
@GetMapping("/books")
public String getBooks(Model model) {
  model.addAttribute("books", bookService.getAll());
  return "books";
}
```

PROJECT ACTIVITIES [0.5H]

Decide architecture for your application. Will you use React or plain JavaScript? Will it be a monolith app or a decoupled application? P.S. The suggested way is to create a decoupled application, start with plain JavaScript on the frontend and migrate to React at a later point. But you can choose other options, as long as you satisfy the requirements of both courses: IDATA2301 and IDATA2306.

No matter what you choose, you can work on the backend logic implementation, because all the service and repository classes should not be dependent on how the data will be presented!