Projektaufgabe 2
Abgabetermin: Woche 6

Allgemeine Informationen

Diese Regeln gelten sowohl für das erste Projekt als auch für alle folgenden Aufgaben.

1. **Team**
   Bis zur **4. Laborwoche** müssen sich die Studierenden in **Teams von zwei Personen** zusammenschließen.
   Jedes Team erhält ein vom Dozenten vorgeschlagenes Thema.

2. **Projektdurchführung**
   Die Bearbeitung der Aufgaben kann mit Hilfe von **ChatGPT** oder anderen **KI-Tools** erfolgen.
   Die Nutzung solcher Tools wird **ausdrücklich empfohlen**, jedoch müssen die Studierenden beim Präsentationstermin **in der Lage sein, Fragen zu beantworten** und zu zeigen, dass sie **den Code und die Logik des Projekts verstehen**.

3. **Projektabgabe**
   Die Abgabe erfolgt über **GitHub**.
   Das Team ist dafür verantwortlich, dass das **Repository öffentlich** ist und der **Dozent Zugriff** darauf hat.
   Im Repository muss die **Beteiligung beider Teammitglieder** sichtbar sein.
   Außerdem muss das Projekt **auf beiden Computern der Teammitglieder lauffähig** sein, nicht nur auf einem.

4. **Bewertung und Präsentation**
   Um eine Note zu erhalten, müssen die Studierenden ihr Projekt **präsentieren** und **Kenntnis des Codes** nachweisen – entweder durch **Beantwortung von Fragen** oder durch **direkte Änderungen im Code** auf Anfrage des Dozenten.

5. **Fristen**
   Jede **Verspätung** bei der Abgabe wird mit **–2 Punkten pro verspäteter Woche** bestraft.

Projekt 2 – Weboberfläche mit Thymeleaf
(Einfaches CRUD für die Projekteinheiten)

Ziel dieser Phase ist es, die Anwendung um eine **Weboberfläche (View-Schicht)** zu erweitern.
Dazu wird **Thymeleaf**, die in Spring Boot integrierte Template-Engine, verwendet.
Sie erstellen die ersten HTML-Seiten, die die in-Memory-Daten der Anwendung anzeigen und einfache Operationen ermöglichen:

Anzeigen, Erstellen und Löschen.

Sie arbeiten weiter an **derselben Thematik und denselben Entitäten** wie in Projektaufgabe 1.
Die Daten bleiben **im Speicher** gespeichert – es wird **keine Datenbank** oder permanente Persistenz verwendet.

---

1. Notwendige Software

Verwenden Sie dieselben Werkzeuge wie in Projektaufgabe 1:

- **JDK 17** (oder neuer)
- **IntelliJ IDEA** (Community Edition ist ausreichend)
- **Git** und ein **GitHub-Konto** (für Zusammenarbeit und Abgabe)

Weitere Programme, Server oder Datenbanken sind **noch nicht erforderlich.**

---

2. Technische Anforderungen

2.1 Hinzufügen der Thymeleaf-Abhängigkeit

Falls Sie Thymeleaf beim Erstellen des Projekts nicht ausgewählt haben, fügen Sie folgende Abhängigkeit in der Datei **pom.xml** hinzu:

```
<dependency>
    <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2.2 Entitäten

Verwenden Sie die in Projektaufgabe definierten Entitäten (gemäß dem bereitgestellten UML-Diagramm).
Für **jede zentrale Entität Ihres Projekts** (z. B. Student, Route, Ticket usw.) soll die entsprechende Weboberfläche implementiert werden.

2.3 Zu implementierende Operationen

Für **jede Hauptentität** sind folgende Funktionen umzusetzen:

- **GET all** – Anzeige aller vorhandenen Objekte
- **CREATE** – Erstellung eines neuen Objekts über ein Formular
- **DELETE** – Löschen eines Objekts über einen Button in der Liste

Nicht erforderlich sind:

- Bearbeitung / Aktualisierung (edit / update)
- Validierungen oder Fehlermeldungen
- Detailansichten einzelner Objekte

2.4 Web-Controller

Erstellen Sie für jede Entität eine mit **@Controller** annotierte Klasse, die folgende Routen behandelt:

- **GET /<entität>** – zeigt die vollständige Liste an
- **GET /<entität>/new** – zeigt das Formular zur Erstellung an
- **POST /<entität>** – verarbeitet das Formular und erstellt das Objekt
- **POST /<entität>/{id}/delete** – löscht das gewählte Objekt

Verwenden Sie für die Löschoperation die Methode **POST**, um die Komplexität der **DELETE**-Methode zu vermeiden und ungewollte Aktionen auszuschließen.

2.5 Thymeleaf-Templates

Legen Sie im Verzeichnis **src/main/resources/templates/** für jede Entität einen Unterordner an:

```
templates/
  bus/
    index.html
    form.html
  route/
    index.html
    form.html
  passenger/
    index.html
    form.html
```

- **index.html** – zeigt alle Objekte in einer Tabelle an und enthält für jede Zeile einen *Delete*-Button.
- **form.html** – ein einfaches Formular zum Hinzufügen eines neuen Objekts.

Beispiel (Ausschnitt aus der Liste):

```
<tr th:each="item : ${buses}">
    <td th:text="${item.id}">1</td>
    <td th:text="${item.number}">B123</td>
    <td th:text="${item.capacity}">50</td>
    <td>
                <form  th:action="@{'/buses/'  +  ${item.id}  +
'/delete'}" method="post">
            <button type="submit">Delete</button>
        </form>
    </td>
</tr>
```

2.6 Daten im Speicher

Verwenden Sie weiterhin die Repositories und Services aus Projektaufgabe 1.
Eine Persistierung in Dateien oder Datenbanken ist **in dieser Phase nicht vorgesehen.**

---

3. Empfohlene Projektstruktur

```
     src/
       main/java/com/example/<thema>/
         model/
         repository/
         service/
         controller/
           BusController.java
           RouteController.java
           PassengerController.java
       main/resources/
         templates/
           bus/
             index.html
             form.html
           route/
             index.html
             form.html
           passenger/
             index.html
             form.html
```

---

4. Definition of Done

- Beim Aufruf von **/bus**, **/route**, **/passenger** usw. zeigt die Anwendung die im Speicher vorhandenen Listen an.
- Neue Objekte können über das Formular korrekt hinzugefügt werden.
- Der *Delete*-Button entfernt das gewählte Objekt und aktualisiert die Liste.
- Es existieren keine Validierungen, Fehlermeldungen oder zusätzlichen Ansichten.
- Alle Seiten werden mit **Thymeleaf** generiert.

---

5. Prinzipien und Best Practices

- Einhaltung des **MVC-Architekturmusters** und des **Single-Responsibility-Prinzips (SRP)**.
- Web-Controller sollen die Services aufrufen, **nicht direkt die Repositories**.
- Jede HTML-Datei hat einen klaren Zweck (Liste / Formular).
- Sauberer, lesbarer und gut strukturierter Code; Verwendung der Java-Konventionen (*camelCase*, *PascalCase*).