1. **RESTfull Webservice mit Spring-Boot**

   Gegeben sind die Klassen `Book` und `BookService`:

```
public class Book implements Serializable {

  private String isbn;
  private double preis;
  private String titel;
  private String author;

  // Weitere Klassenmitglieder
}
```

   Die statische Methode der Klasse `BookService`

```
public static List<Book> getBooks()
```

   liefert eine Liste von Testdaten.

   Aufgabe

   Erstellen Sie mit Hilfe von Spring-Boot eine REST-Resource mit folgender Funktionalität:

   - Mit Hilfe dieser GET-Methode sollen alle gespeicherten Bücher angefragt werden können.

     `URI: /books`

   - Mit Hilfe dieser GET-Methode können alle Bücher angefragt werden, in deren Titel der String `pattern` (z.B. SQL) vorkommt.

     `URI: /books/{pattern}`

   Es genügt, wenn Sie die Funktionalität mit Hilfe eines Webbrowsers testen.

2. **REST-Endpunkt mit SpringBoot**

   Mit Hilfe von SpringBoot ist ein Restendpunkt für eine einfache SocialMedia-App zu realisieren. Dabei sind zwei Entities, `User` und `Post` zu realisieren.

   Die beiden Entitäten haben den folgenden Aufbau:

```
public class User implements Serializable {

  private Integer id;          // Autogenerierter PK
  private String name;         // Nickname des Users
  private LocalDate birthday;  // Geburtstag

  // Weitere Klassenmitglieder
}

public class Post implements Serializable {
```

```
  private Integer id;          // Autogenerierter PK
  private String message;      // Nachricht

  // weitere Klassenmitglieder

}
```

Dabei sind folgende Vorgaben zu beachten:

- Der Username soll unique sein und aus wenigstens 3 Zeichen bestehen.
- Der Geburtstag des Users darf nicht in der Zukunft liegen.
- Die Nachricht im Post darf nicht leer sein.
- Die 1:n - Beziehung zwischen User und Post ist bidirektional zu mappen.

Die Restschnittstelle soll wenigstens die folgenden Endpunktmethoden zur Verfügung stellen:

```
- Abfrage aller User               - GET /users
- Speichern eines neuen Users      - POST /users
- Abfrage eines bestimmten Users   - GET /users/{id}
- Löschen eines Users              - DELETE /users/{id}

- Abfrage aller Posts eines Users  - GET /users/{id}/posts
- Speicheren eines neuen Posts     - POST /users/{id}/posts
- Abfrage eines bestimmten Posts   - GET /posts/{id}
```

Beim Löschen eines Users sollen auch alle Posts des Users gelöscht werden.

Bei der Implementierung des Microservice sind folgende Vorgaben zu beachten:

- Die Erzeugungsstrategie ist auf create-drop zu setzen und beim Starten der Applikation sind wenigstens zwei User und zu jedem User 3 Posts zu erzeugen.
- Es ist eine H2 InMemory-Datenbank zu verwenden.
- Alle allfälligen Fehler sind über einen zentralen Exceptionhandler zu behandeln.
- Implementieren Sie die oben angegebenen Validierungen und reagieren Sie angemessen auf Validierungsfehler.

Testen Sie Ihre Endpunktmethoden mit Postman. Abzugeben ist auch ein Protokoll, das die von Ihnen durchgeführten Tests mit den erwarteten Ergebnissen auflistet.

Hinweis

Übernimmt man ein Datum im ISO-Format in eine Pfadvariable vom Typ `LocalDate`, so muss der String wie folgt in ein LocalDate konvertiert werden:

```
@GetMapping("/anyrote/{date}")
public String handleAnyRoute(@PathVariable("startdate")
   @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate date) {
// method body
}
```