Laboratorium 1 – REST

Celem ćwiczenia jest przygotowanie usługi internetowej zgodnej ze stylem architektonicznym REST (ang. RESTful web service/RESTful Web API). Usługę należy zaimplementować w języku Java z wykorzystaniem standardu JAX-RS i jego referencyjnej implementacji Jersey oraz serwera HTTP Grizzly. Możesz wykorzystać dokumentację Jersey (zobacz przykład użycia Jersey z Grizzly). Do testowania usługi polecany jest Postman.

Grupa zadań A (1p):

1. Utwórz w IntelliJ nowy projekt typu Maven. Pamiętaj o wpisaniu w pom.xml informacji o używanej wersji Javy oraz dopisz zależność do bibliotek Grizzly, Jersey oraz ich zależności (można wykorzystać nowsze wersje niż w przykładzie):

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.5.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>

<dependencies>
    <dependency>
        <groupId>org.glassfish.jersey.core</groupId>
        <artifactId>jersey-server</artifactId>
        <version>2.22.2</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-json-jackson</artifactId>
        <version>2.22.2</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.ext</groupId>
        <artifactId>jersey-declarative-linking</artifactId>
        <version>2.22.2</version>
    </dependency>
    <dependency>
        <groupId>javax.el</groupId>
        <artifactId>javax.el-api</artifactId>
        <version>2.2.4</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.web</groupId>
        <artifactId>javax.el</artifactId>
        <version>2.2.4</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.containers</groupId>
        <artifactId>jersey-container-grizzly2-http</artifactId>
        <version>2.22.2</version>
```

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

2. Przygotuj model danych reprezentujący następujące zasoby:
    a. Student – posiada numer indeksu (unikalny identyfikator), imię, nazwisko, data urodzenia (typ Date); informacje nie mogą być puste,
    b. Przedmiot – posiada nazwę, prowadzącego i kolekcję ocen,
    c. Ocena – obiekt zawierający wartość liczbową (2.0 – 5.0, co 0.5), datę wystawienia i przypisanego studenta (referencja na obiekt).

    Pamiętaj, że może istnieć wiele studentów, przedmiotów oraz ocen studenta z przedmiotu! Każda z klas modelu musi mieć adnotację `@XmlRootElement`, bezargumentowy konstruktor oraz gettery i settery pól w celu jej automatycznej (de-)serializacji. Wypełnij utworzony model kilkoma krotkami danych (dowolnych).

Grupa zadań B (1p):

3. Przygotuj interfejs REST eksponujący w/w model danych nadając im hierarchiczną adresację (np.: ocena jest podzasobem przedmiotu lub studenta), umożliwiający następujące operacje na zasobach:
    a. GET – pobranie kolekcji studentów/przedmiotów/ocen z przedmiotu
    b. GET – pobranie pojedynczego studenta/przedmiotu/oceny
    c. POST – dopisanie studenta/przedmiotu/oceny do modelu
    d. PUT – aktualizacja studenta/przedmiotu/oceny
    e. DELETE – usunięcie studenta/przedmiotu/oceny
    f. Pamiętaj, że kod odpowiedzi HTTP powinien odpowiadać zrealizowanej operacji (czyli np.: 201 zamiast 200 po utworzeniu zasobu)
    g. Pamiętaj, że zwrócenie pewnych kodów HTTP wiąże się z koniecznością zwrócenia odpowiedniego nagłówka (np.: dla 201 nagłówek `Location`).
4. Udostępnij w/w zasoby w reprezentacjach: XML oraz JSON oraz wykorzystaj mechanizm HTTP negocjacji zawartości (nagłówek `Accept`) w celu przesłania klientowi żądanej reprezentacji. Użyj mechanizmów serializacji/deserializacji obiektów dostępnych w JAXB/JAX-RS.