Laboratorium 3 – 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 E (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 bibliotek niż w przykładzie):

```
<properties>
    <maven.compiler.source>1.11</maven.compiler.source>
    <maven.compiler.target>1.11</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>org.glassfish.jersey.core</groupId>
        <artifactId>jersey-server</artifactId>
        <version>2.28</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-json-jackson</artifactId>
        <version>2.28</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.ext</groupId>
        <artifactId>jersey-declarative-linking</artifactId>
        <version>2.28</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.core</groupId>
        <artifactId>jersey-common</artifactId>
        <version>2.28</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.inject</groupId>
        <artifactId>jersey-hk2</artifactId>
        <version>2.28</version>
    </dependency>
    <dependency>
        <groupId>javax.xml.bind</groupId>
        <artifactId>jaxb-api</artifactId>
        <version>2.3.1</version>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.bind</groupId>
        <artifactId>jaxb-core</artifactId>
        <version>2.3.0.1</version>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.bind</groupId>
        <artifactId>jaxb-impl</artifactId>
        <version>2.3.1</version>
```

```
        </dependency>
        <dependency>
            <groupId>javax.el</groupId>
            <artifactId>javax.el-api</artifactId>
            <version>3.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.web</groupId>
            <artifactId>javax.el</artifactId>
            <version>2.2.6</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.jersey.containers</groupId>
            <artifactId>jersey-container-grizzly2-http</artifactId>
            <version>2.28</version>
        </dependency>
    </dependencies>
```

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

    Pamiętaj, że może istnieć wiele studentów, przedmiotów oraz ocen studenta z każdego 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).
4. 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 studenta
    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)

Grupa zadań F (1p):

5. 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.
6. Uzupełnij usługę tak, aby działała zgodnie z 3. poziomem dojrzałości Richardsona (HATEOAS) – dodaj łącza do powiązanych zasobów w odpowiedzi HTTP. Użyj adnotacji @InjectLink. Przykładowe poprawne zastosowanie @InjectLink zaprezentowano poniżej (pole w klasie).

```
@InjectLinks({
      @InjectLink(resource = resources.Student.class, rel = "self"),
      @InjectLink(resource = resources.Students.class, rel = "parent"),
      @InjectLink(resource = resources.Grades.class, rel = "grades")
})
@XmlElement(name="link")
```

```
@XmlElementWrapper(name = "links")
@XmlJavaTypeAdapter(Link.JaxbAdapter.class)
List<Link> links;
```

7. Opcjonalnie: Wykorzystaj mechanizm HTTP Authentication do ograniczenia dostępu do wybranych zasobów, w taki sposób, że:
    a. Prowadzący może modyfikować zasoby (np.: aktualizować opis przedmiotu, dopisać/usunąć ocenę itp.)
    b. Prowadzący może przeglądać wszystkie oceny wszystkich studentów
    c. Student może przeglądać oceny wszystkich studentów

    Wykorzystaj mechanizm javax.annotation.security i dostęp do zasobów bazujący na rolach.
8. Opcjonalnie: Utwórz drugi moduł w ramach tego samego projektu, reprezentujący klienta w/w usługi. Aplikacja powinna wykorzystywać technologię JAX-RS do połączenia się z usługą, pobrania i/lub manipulacji zasobami. Aplikacja może działać zarówno w kontekście studenta, jak i prowadzącego. Załóż, że aplikacja otrzymuje, jako argumentu wyłącznie główny adres usługi, łącza do dostępnych zasobów pobierane są z odpowiedzi serwera (HATEOAS). Wystarczający jest interfejs konsolowy.