RESTful Web Services na platformie Java EE / Jakarta EE (JAX-RS)

Do realizacji ćwiczenia wymagane jest środowisko NetBeans 17 wraz z serwerem aplikacji Payara i dostarczanym wraz z nim wbudowanym serwerem bazy danych H2. Potrzebne będzie również narzędzie do testowania API dostępnego poprzez protokół HTTP. W opisie ćwiczenia przyjęto, że narzędziem tym będzie Postman.

Ćwiczenie 1

Celem ćwiczenia jest przygotowanie usługi sieciowej opartej na klasie Java oznaczonej adnotacjami.

1. Uruchom NetBeans i utwórz nowy projekt opcją File→New Project... W kreatorze projektu z listy kategorii wybierz Java with Maven, a z listy projektów wybierz Web Application. Kliknij przycisk Next >. Jako nazwę projektu podaj **Complaints**, wyczyść pole z nazwą pakietu i kliknij przycisk Next >. Wybierz wersję Java EE Jakarta EE 10 Web i upewnij się, że jako serwer aplikacji wybrany jest serwer Payara (jeśli nie jest dostępny do wyboru, to kliknij Add... i pobierz oraz zainstaluj najnowszą dostępną wersję). Kliknij przycisk Finish.

   Odszukaj w pliku `pom.xml` wersję maven-war-plugin. Jeśli podana jest wcześniejsza niż 3.3.2, to zmień ją na 3.3.2. Zapisz zmiany.

2. Odszukaj w strukturze projektu plik `persistence.xml` (powinien zostać utworzony przez kreator projektu i znajdować się w podkatalogu META-INF). Przejdź do edycji jego źródła otwierając plik, a następnie przełączając edytor na zakładkę Source.

3. Zastąp w kodzie źródłowym pliku `persistence.xml` zawartość elementu <persistence-unit> poniższą wersją, specyfikującą jednostkę trwałości powiązaną ze źródłem danych na serwerze aplikacji i wykorzystującą transakcje w standardzie JTA.

```
<jta-data-source>jdbc/__default</jta-data-source>
<exclude-unlisted-classes>false</exclude-unlisted-classes>
<properties>
  <property
   name="jakarta.persistence.schema-generation.database.action"
   value="create"/>
</properties>
```

4. Utwórz klasę encji `Complaint` do reprezentowania skarg.

   a. Kliknij prawym przyciskiem myszy na ikonie projektu i z menu kontekstowego wybierz New → Entity Class.

   b. Jako nazwę klasy podaj `Complaint`, a jako nazwę pakietu `entities`. Pozostaw zaproponowany typ `Long` jako typ identyfikatora encji.

   c. Popraw nazwy importowanych pakietów (`javax` => `jakarta`).

   d. Dodaj w klasie encji następujące prywatne pola: `complaintDate` typu `LocalDate`, `complaintText` typu `String`, `author` typu `String` oraz

`status` typu `String`. Dodaj import klasy `java.time.LocalDate`. Wygeneruj publiczne gettery i settery dla dodanych pól (skorzystaj z kreatora Refactor → Encapsulate Fields).

5. Dodaj poniższe ograniczenia dla pól encji `Complaint` za pomocą adnotacji Bean Validation:
   a. @NotNull dla wszystkich 4 dodanych pól
   b. @Size(min = 1, max = 60) dla pól `complaintText` i `author`
   c. @Size(min = 1, max = 6) dla pola `status`

6. Utwórz w projekcie klasę `EntityManagerProducer`, który będzie implementować bean CDI pozwalający uzyskać obiekt `EntityManager` w sposób pozwalający na jego wstrzyknięcie jak beana CDI z jednoczesnym zapewnieniem, że nie będzie on współdzielony przez różne żądania. Umieść ją w pakiecie `data`. W tym celu:
   a. Z menu kontekstowego projektu uruchom kreator Java Class z kategorii Java.
   b. Wprowadź nazwę klasy i nazwę pakietu, po czym kliknij przycisk Finish.
   c. Jako ciało klasy wklej poniższy kod. Zwróć uwagę na zwykłe wstrzyknięcie obiektu `EntityManager` i na metodę producenta, która „opakowuje" go jako bean CDI o zasięgu żądania.

```
@PersistenceContext
private EntityManager em;

@Produces
@RequestScoped
public EntityManager getEntityManager() {
    return em;
}
```

   d. Wstaw poniższe instrukcje `import`. (Uwaga: W tym wypadku należy zwrócić uwagę by zaimportować adnotacje z właściwych pakietów, gdyż niektóre z nich powtarzają się w różnych pakietach.)

```
import jakarta.enterprise.context.RequestScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
```

7. Utwórz w projekcie klasę `ComplaintRepository`, który będzie oferować funkcjonalność CRUD do obsługi encji `Complaint`. Umieść ją w pakiecie `data`. W tym celu:
   a. Z menu kontekstowego projektu uruchom kreator Java Class z kategorii Java.
   b. Wprowadź nazwę interfejsu i nazwę pakietu, po czym kliknij przycisk Finish.
   c. Oznacz klasę adnotacją `@ApplicationScoped`.

d. W ciele klasy umieść poniższą kod wstrzykujący obiekt `EntityManager` jako bean CDI.

```
@Inject
private EntityManager em;
```

Komentarz: Zalecany zasięg CDI dla beana implementującego funkcjonalność repozytorium to zasięg aplikacyjny, który sprawi że jedna instancja beana będzie współdzielona przez wszystkie żądania i sesje użytkowników. W przypadku gdyby do takiego beana `EntityManager` był wstrzyknięty zwyczajnie adnotacją `@PersistenceContext`, to z racji zasięgu aplikacyjnego beana repozytorium, byłby on również współdzielony przez wszystkie żądania, co stanowiłoby duży błąd, gdyż obiekty `EntityManager` nie są „thread-safe". W przyjętym rozwiązaniu `EntityManager` jest „produkowany" z zasięgiem pojedynczego żądania przez metodę producenta zawartą w beanie o domyślnym zasięgu `@Dependent`, co jest odpowiednie w tym wypadku. CDI obsługuje wstrzykiwanie beana o mniejszym zasięgu do beana o większym zasięgu (co ma miejsce w naszym rozwiązaniu) poprzez obiekty proxy.

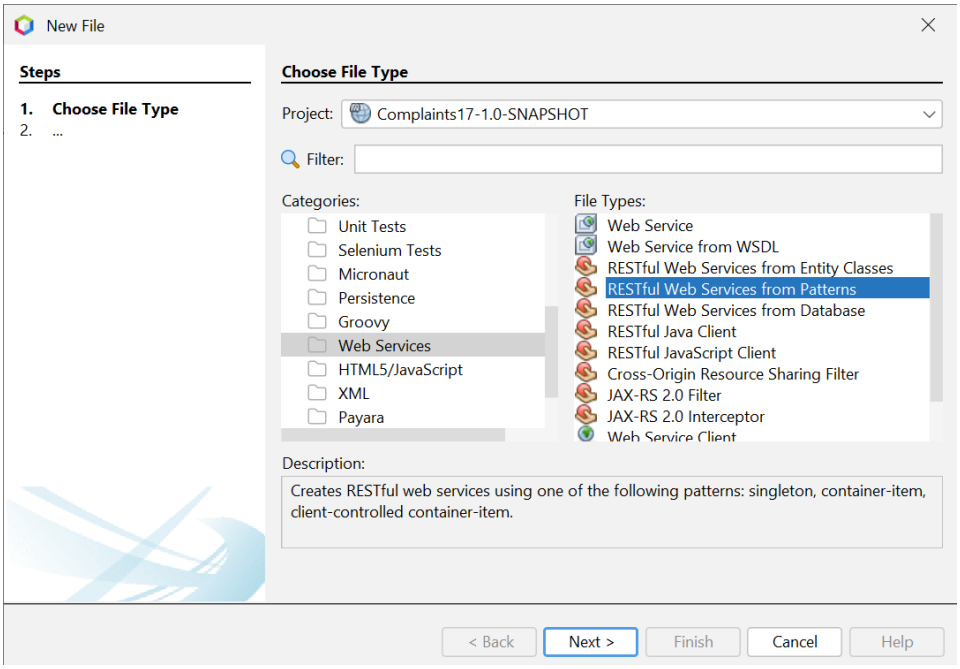
e. Umieść w ciele klasy poniżej wstrzyknięcia obiektu `EntityManager` poniższe metody repozytorium.

```
public void create(Complaint entity) {
    em.persist(entity);
}

public void edit(Complaint entity) {
    em.merge(entity);
}

public void remove(Complaint entity) {
    em.remove(em.merge(entity));
}

public Complaint find(Object id) {
    return em.find(Complaint.class, id);
}

public List<Complaint> findAll() {
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    cq.select(cq.from(Complaint.class));
    return em.createQuery(cq).getResultList();
}
```

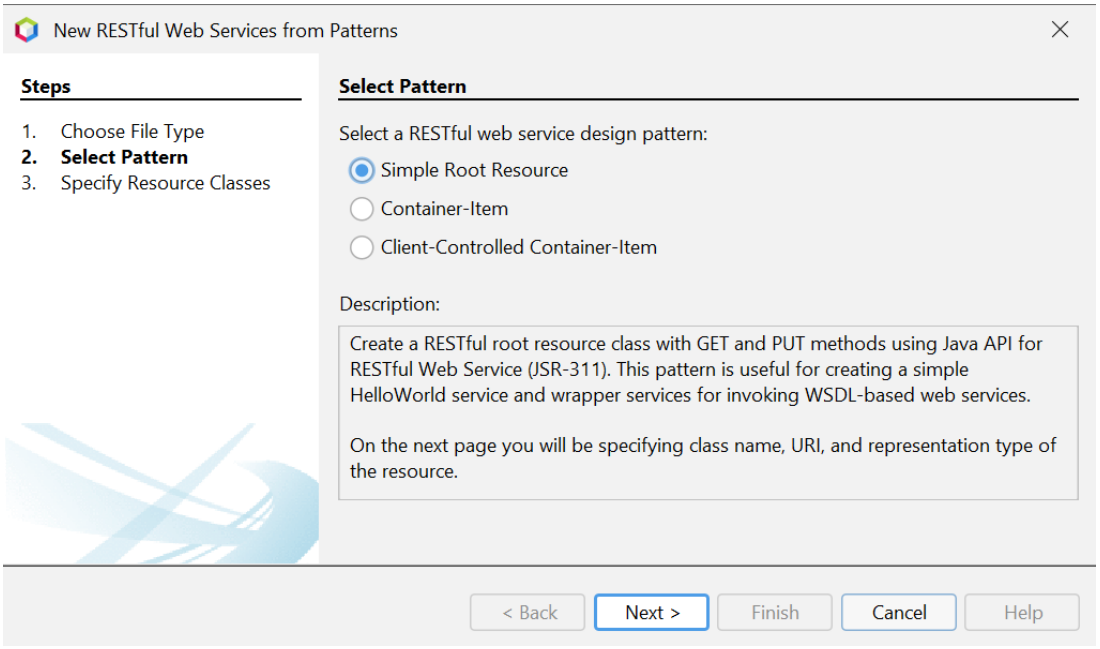
f. Uzupełnij brakujące importy. Zwróć uwagę aby klasy/interfejsy JPA importować z pakietów o nazwach rozpoczynających się od „jakarta".

8. Zanim utworzymy usługę REST obsługującą skargi, stworzymy klasę DTO je reprezentującą aby nie wykorzystywać bazodanowej encji w warstwie zasobów. Utwórz więc w projekcie klasę `ComplaintDTO`, umieszczając ją w pakiecie `dto`. Klasa DTO powinna zawierać takie same pola jak encja, wraz z getterami i setterami, z adnotacjami Bean Validation, ale bez adnotacji JPA.

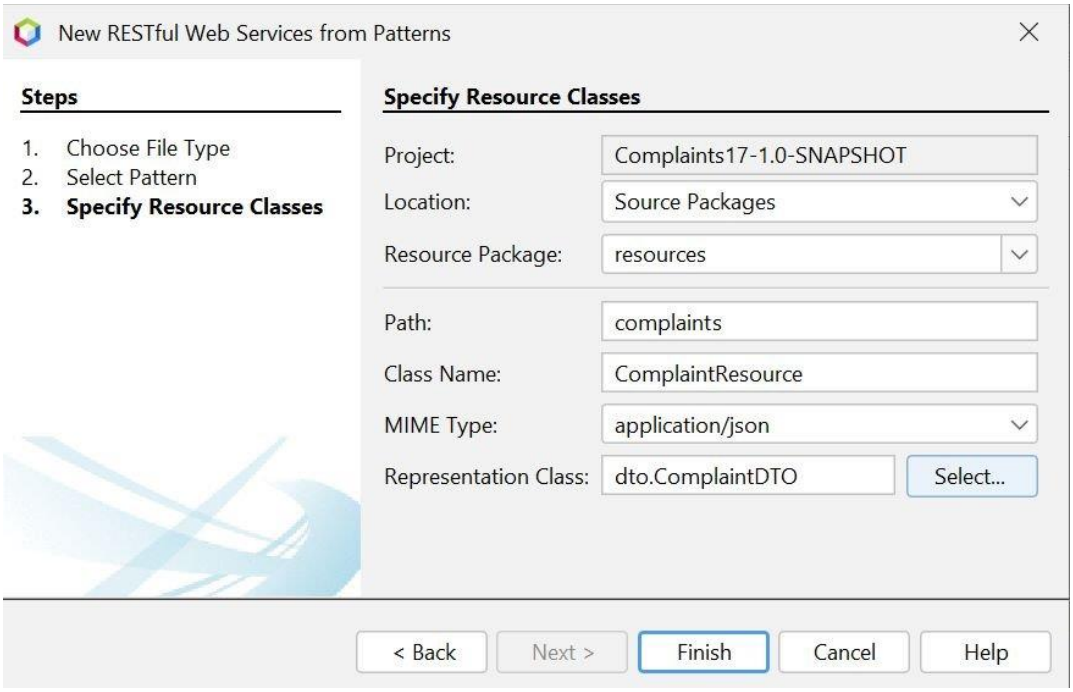
9. Uruchom w projekcie kreator RESTful Web Services from Patterns z kategorii Web Services.



10. W drugim kroku kreatora jako wzorzec projektowy wybierz Simple Root Resource i przejdź do kolejnego kroku.



11. W ostatnim kroku kreatora zmień ustawienia jak na poniższym obrazku:



12. Otwórz wygenerowaną klasę zasobu i popraw nazwy pakietów w importach (`javax => jakarta`).
13. Otwórz wygenerowaną klasę konfiguracyjną usługi REST i popraw nazwę pakietu w adnotacji opisującej klasę (`javax => jakarta`). Od razu popraw też ścieżkę w adnotacji na krótszą: „resources”.
14. Zanim dokończymy implementację zasobu REST, utworzymy jeszcze pomocniczą klasę dla logiki biznesowej. Dzięki niej klasa zasobu nie będzie zajmowała się przetwarzaniem transakcyjnym i konwersjami między DTO a encją, co pozwoli zachować w projekcie zasadę pojedynczej odpowiedzialności i niezależność warstw aplikacji od siebie. Utwórz więc w projekcie klasę `ComplaintService`, umieszczając ją w pakiecie `services`.
15. Klasa `ComplaintService` w naszej aplikacji nie będzie zawierała skomplikowanej logiki biznesowej. Będzie ona zawierała metody odpowiadające metodom repozytorium i je wywołujące. Ponieważ repozytorium operuje na encji, klasa logiki biznesowej będzie musiała dokonywać konwersji encji na DTO i w drugą stronę. Do konwersji wykorzystamy bibliotekę ModelMapper. Dodaj ją do zależności projektu w pliku `pom.xml`:

```
<dependency>
  <groupId>org.modelmapper</groupId>
  <artifactId>modelmapper</artifactId>
  <version>3.0.0</version>
</dependency>
```

16. Wróć do edycji klasy `ComplaintService`.
    a. Dodaj następujące importy:

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;
import data.ComplaintRepository;
import dto.ComplaintDTO;
```

```
import entities.Complaint;
import org.modelmapper.ModelMapper;
import org.modelmapper.TypeToken;
import java.util.List;
import java.lang.reflect.Type;
import jakarta.transaction.Transactional;
```

a. Oznacz klasę adnotacją CDI `@ApplicationScoped`.

b. Wstrzyknij do niej repozytorium.

c. Zaimplementuj metody create(), edit(), remove() i find() wzorując się na poniższej implementacji create().

```
public void create(ComplaintDTO dto) {
    ModelMapper mapper = new ModelMapper();
    repository.create(mapper.map(dto, Complaint.class));
}
```

d. Dodaj poniższą implementację findAll(). W tym wypadku konwersja jest bardziej skomplikowana, gdyż dotyczy ona list generycznych.

```
public List<ComplaintDTO> findAll() {
    ModelMapper mapper = new ModelMapper();
    List<Complaint> entityList = repository.findAll();
    Type listType =
         new TypeToken<List<ComplaintDTO>>() {}.getType();
    List<ComplaintDTO> dtoList =
                       mapper.map(entityList, listType);
    return dtoList;
}
```

e. Oznacz wszystkie metody klasy z logiką biznesową adnotacją `@Transactional`.

17. Wróć do edycji klasy zasobu i dokończ jej implementację:

a. Wstrzyknij komponent logiki biznesowej.

b. Usuń metody dodane przez kreator i zastąp je poniższym zestawem operacji CRUD:

```
@GET
@Produces(jakarta.ws.rs.core.MediaType.APPLICATION_JSON)
public List<ComplaintDTO> getAllComplaints() {
    return service.findAll();
}

@GET
@Path("{id}")
@Produces(jakarta.ws.rs.core.MediaType.APPLICATION_JSON)
public ComplaintDTO getComplaint(@PathParam("id") Long id) {
    return service.find(id);
}

@POST
@Consumes(jakarta.ws.rs.core.MediaType.APPLICATION_JSON)
public void postComplaint(ComplaintDTO complaint) {
    service.create(complaint);
}
```

```
    @PUT
    @Path("{id}")
    @Consumes(jakarta.ws.rs.core.MediaType.APPLICATION_JSON)
    public void putComplaint(@PathParam("id") Long id, ComplaintDTO
complaint) {
        service.edit(complaint);
    }

    @DELETE
    @Path("{id}")
    public void deleteComplaint(@PathParam("id") Long id) {
        service.remove(service.find(id));
    }
```

18. Uruchom aplikację.
19. Sprawdź z poziomu paska adresu przeglądarki reakcję aplikacji na żądanie GET pobierające wszystkie skargi.
20. Z poziomu paska adresu przeglądarki można przetestować odpowiedzi API na żądania GET. Aby przetestować reakcję na inne metody HTTP należy wykorzystać dedykowane do tego celu narzędzia lub samodzielnie zaimplementować aplikacje klienckie. W ćwiczeniu wykorzystamy narzędzie Postman. Uruchom je i na początek przetestuj to samo żądanie GET pobierające wszystkie skargi.
21. Przetestuj w narzędziu Postman możliwość tworzenia nowych instancji skarg:
    a. Wprowadź odpowiedni URI
    b. Wybierz metodę POST
    c. Upewnij się, że typem przesyłanej zawartości jest JSON
    d. Jako ciało żądania (w trybie „raw”) wprowadź:

```
{
    "author": "Jim Brown",
    "complaintDate": "2021-04-22",
    "complaintText": "Please check TV in room 242",
    "status": "closed"
}
```

    e. Zwróć uwagę na kod statusu odpowiedzi HTTP. Co on oznacza?
22. Przełącz się na okno przeglądarki i pobierz wszystkie skargi. Powinna zostać pobrana tablica JSON ze skargą dodaną przed chwilą z poziomu narzędzia Postman.
23. Wróć do narzędzia Postman i dodaj poniższe skargi tym samym sposobem co poprzednio:

```
{
    "author": "Marvin Doherty",
    "complaintDate": "2021-04-23",
    "complaintText": "Please fix a tap in room 234",
    "status": "open"
}
```

```
{
    "author": "Arthur McCoy",
    "complaintDate": "2021-04-24",
    "complaintText": "Repair fridge in room 311",
    "status": "open"
}
```

```
{
    "author": "Jim Brown",
    "complaintDate": "2021-04-24",
    "complaintText": "Remove the blood stains on the
wall in room 242",
    "status": "open"
}
```

```
{
    "author": "Johny Bravo",
    "complaintDate": "2021-04-24",
    "complaintText": "Fix air conditioning in room
242",
    "status": "open"
}
```

24. Pobierz wszystkie skargi z poziomu Postmana i przeglądarki.
25. Pobierz z poziomu Postmana i przeglądarki skargę o podanym identyfikatorze. (Identyfikatory zostały nadane automatycznie w bazie danych. Odczytaliśmy je wraz z resztą informacji o skargach w poprzednim punkcie.)
26. Usuń z poziomu Postmana jedną ze skarg o statusie „open”. Zwróć uwagę na kod statusu odpowiedzi HTTP. Następnie pobierz wszystkie skargi aby upewnić się, że usunięcie faktycznie się powiodło.
27. Dodaj możliwość sprawdzenia statusu skargi poprzez adres URI.
    a. Otwórz klasę zasobu.
    b. Dodaj metodę `checkStatus` o następującej treści:

```
public String checkStatus(Long id) {
    return service.find(id).getStatus();
}
```

    c. Metoda ta musi być dostępna przez wywołanie GET. Dodaj odpowiednią adnotację.
    d. Ustaw ścieżkę, pod którą dostępna będzie ta metoda, na „{id}/status”. Ponownie dodaj stosowną adnotację.
    e. Ostatnią adnotacją dla metody zapewnij by status udostępniony był czystym tekstem.

f. Stwórz powiązanie między parametrem „id” w nagłówku metody a polem „{id}” w jej adresie. Wykorzystaj do tego adnotację @PathParam.

g. Uruchom aplikację i przetestuj w przeglądarce odczyt statusu dla kilku skarg.

28. Z poziomu Postmana zaktualizuj jedną ze skarg o statusie „open”, modyfikując coś w jej treści i zmieniając status na „closed”. W tym celu wyślij żądanie metodą PUT pod adres zawierający identyfikator wybranej skargi, podając jako treść żądania odpowiedni JSON. Następnie sprawdź żądaniem GET czy modyfikacja została zrealizowana.

29. Dodaj obsługę filtrowania skarg według statusu:

    a. Oznacz klasę encji przedstawioną poniżej adnotacją `@NamedQuery` definiującą nazwane zapytanie JPQL do wyboru skarg o podanym stutusie.

    ```
    @NamedQuery(name = "Complaint.findByStatus", query =
    "SELECT c FROM Complaint c WHERE c.status = :status"
    )
    ```

    b. W klasie zasobu do metody `getAllComplaints` dodaj parametr typu String o nazwie `status`.

    c. Adnotacją `@QueryParam` powiąż go z nazwą parametru query string „status”. Uwaga: Wcześniej wykorzystywaliśmy parametry ścieżkowe. Parametry ścieżkowe są odpowiednie do identyfikacji zasobu. Do filtracji lub sortowania zalecane jest używanie parametrów zawartych w łańcuchu query string. Zwróć uwagę, że dla parametrów typu query nie trzeba zmieniać ścieżki związanej z metodą klasy.

    d. Przekaż wartość dodanego parametru do wywoływanej metody komponentu logiki biznesowej, a z niej z kolei do metody repozytorium.

    e. Dodaj do metody `findAll` repozytorium kod, który zwróci dotychczasowy rezultat, jeżeli parametr `status` jest pusty (`null`), a w przeciwnym wypadku zwróci wynik wywołania zapytania nazwanego (`NamedQuery`) „Complaint.findByStatus” przekazując do niego wartość parametru `status`.

```
if (status != null && !"".equals(status))
  return em.createNamedQuery("Complaint.findByStatus")
           .setParameter("status", status)
           .getResultList();
else
  ...
```

30. Uruchom usługę i przetestuj Postmanem działanie filtrowania wg statusu dla zasobu `complaints`.

31. Przetestuj filtrowanie skarg wg statusu bezpośrednio wprowadzając odpowiedni adres w pasku adresu przeglądarki (bez pomocy Postmana).

Ćwiczenie 2

Celem ćwiczenia jest refaktoryzacja w celu współdzielenia instancji `ModelMapper`. Aktualnie każda metoda komponentu logiki biznesowej tworzy nową instancję `ModelMapper` w pierwszym kroku swojego działania. Gdy wszędzie w aplikacji ModelMapper jest używany w tej samej konfiguracji (a tak jest w naszym przypadku, gdyż używamy zawsze domyślnej konfiguracji), lepiej jest współdzielić instancję, ponieważ są one thread-safe. W aplikacji wykorzystującej CDI preferowanym sposobem współdzielenia instancji `ModelMapper` jest opakowanie jej beanem CDI o zasięgu `@ApplicationScoped`.

1. Utwórz w pakiecie service klasę `ModelMapperProducer`.
2. Zaimplementuj w niej metodę, która tworzy instancję `ModelMapper` i ją zwraca jako wynik. Oznacz tę metodę adnotacjami `@Produces` i `@ApplicationScoped`.
3. Wstrzyknij `ModelMapper` do komponentu logiki biznesowej.
4. Usuń niepotrzebne już operacje tworzenia instancji `ModelMapper` w poszczególnych metodach komponentu logiki biznesowej, w zamian wykorzystując wstrzykniętą instancję.
5. Sprawdź czy po refaktoryzacji aplikacja nadal działa poprawnie.

Ćwiczenie 3

Celem ćwiczenia jest przygotowanie klienta w formie konsolowej aplikacji Java dla usługi REST utworzonej w pierwszym ćwiczeniu.

1. Utwórz w środowisku NetBeans nowy projekt typu Java Application z kategorii Java with Maven. Kliknij przycisk Next >. Jako nazwę projektu podaj **ComplaintsClient** i kliknij przycisk Finish.
2. Utwórz w utworzonym przed chwilą projekcie klasę `app.Main`.
3. Dodaj w pliku `pom.xml` projektu klienta bibliotekę Jersey (implementację JAX-RS):

```
    <dependencies>
        <dependency>
            <groupId>org.glassfish.jersey.core</groupId>
            <artifactId>jersey-client</artifactId>
            <version>3.1.1</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.jersey.inject</groupId>
            <artifactId>jersey-hk2</artifactId>
            <version>3.1.1</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-moxy</artifactId>
            <version>3.1.1</version>
        </dependency>
    </dependencies>
```

4. Utwórz w klasie `app.Main` metodę `main()` z poniższym kodem, w miejsce <id> wpisując jeden z istniejących w bazie identyfikatorów skarg:

```
Client client = ClientBuilder.newClient();
String status =
client.target("http://localhost:8080/Complaints/" +
              "resources/complaints/<id>/status")
.request(MediaType.TEXT_PLAIN)
.get(String.class);

System.out.println("Status: " + status);

client.close();
```

   Zaimportuj wykorzystywane klasy/interfejsy z pakietów `jakarta.ws.rs.client` i `jakarta.ws.rs.core`.

5. Uruchom aplikację klienta.
6. Samodzielnie (możesz wzorować się na przykładach np. z Java EE / Jakarta EE Tutorial) dodaj w metodzie `main()` klienta następujące operacje i przetestuj ich działanie.
   a. Pobierz i wyświetl na konsoli wszystkie skargi.
   b. Pobierz i wyświetl na konsoli jedną z otwartych skarg (przesyłając jej identyfikator do usługi).
   c. Zmodyfikuj skargę pobraną w poprzednim punkcie zmieniając jej status na zamknięty (poprzez podmianę całego zasobu).
   d. Pobierz i wyświetl na konsoli wszystkie otwarte skargi.