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

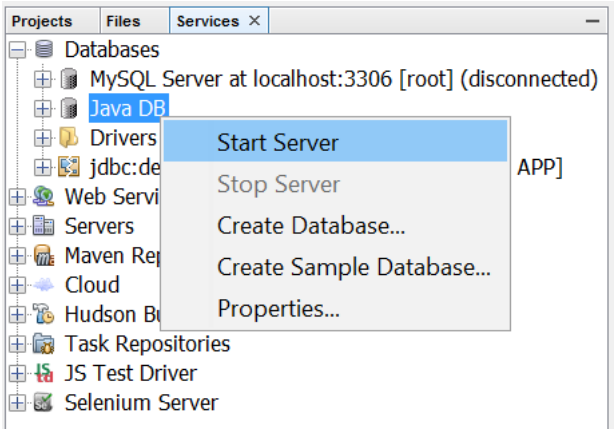
Do realizacji ćwiczenia wymagane jest środowisko NetBeans 11.x wraz z serwerem aplikacji GlassFish i serwerem bazy danych Java DB. Potrzebne będzie również narzędzie do testowania API dostępnego poprzez protokół HTTP. W opisie ćwiczenia przyjęto, że narzędziem tym będzie Postman.

Ćwiczenie 1

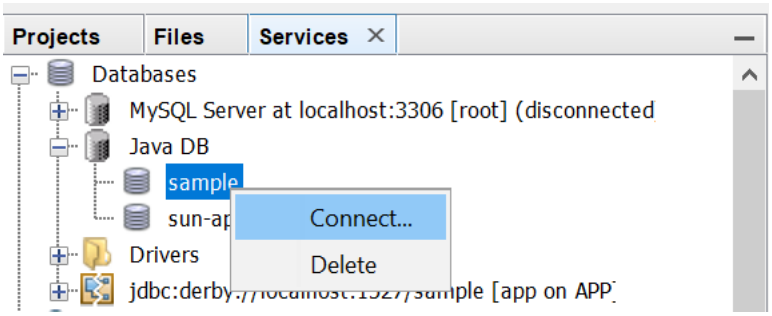
Celem ćwiczenia jest przygotowanie usługi sieciowej opartej na klasie Java oznaczonej adnotacjami.

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 Java EE 8 Web i upewnij się, że jako serwer aplikacji wybrany jest GlassFish (jeśli nie jest dostępny do wyboru, to kliknij Add... i pobierz oraz zainstaluj najnowszą dostępną wersję). Kliknij przycisk Finish.

2. Uruchom serwer bazy danych Java DB korzystając z panelu Services.



3. Z poziomu panelu Services połącz się z bazą danych „sample” na lokalnym serwerze Java DB (derby).



4. Utwórz w bazie danych tabelę, na której działać będzie tworzona aplikacja, realizując poniższe kroki:

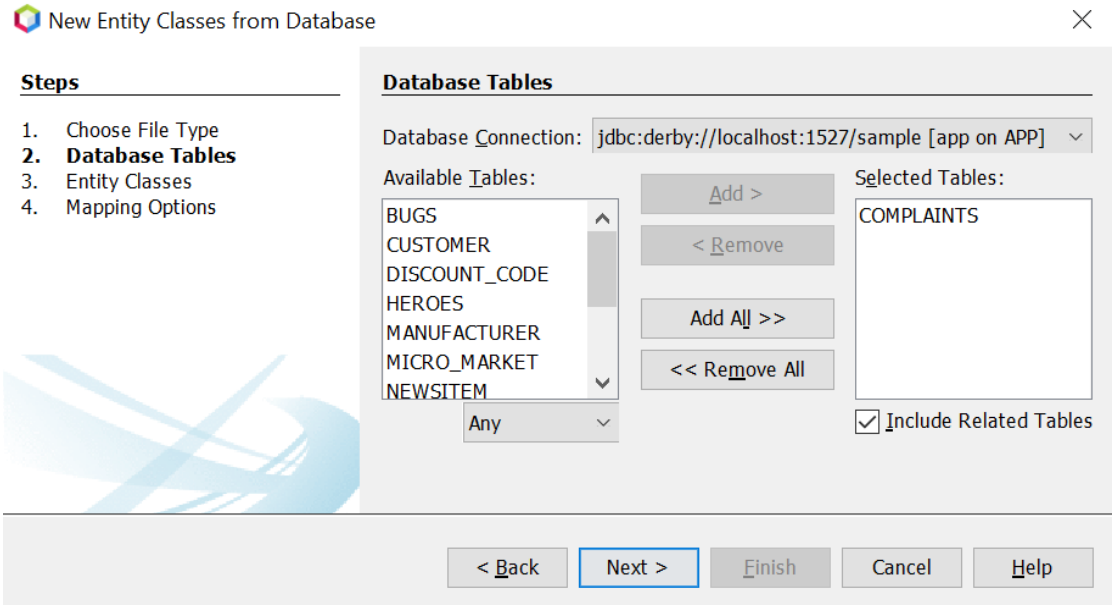
    a. Wywołaj okno poleceń SQL dla bazy danych „sample” wybierając z menu kontekstowego dla węzła reprezentującego połączenie opcję Execute Command.

    b. Korzystając z okna poleceń SQL wykonaj kolejno poniższe polecenia SQL (pierwsze może zakończyć się błędem – służy do ewentualnego usunięcia istniejącej już tabeli):

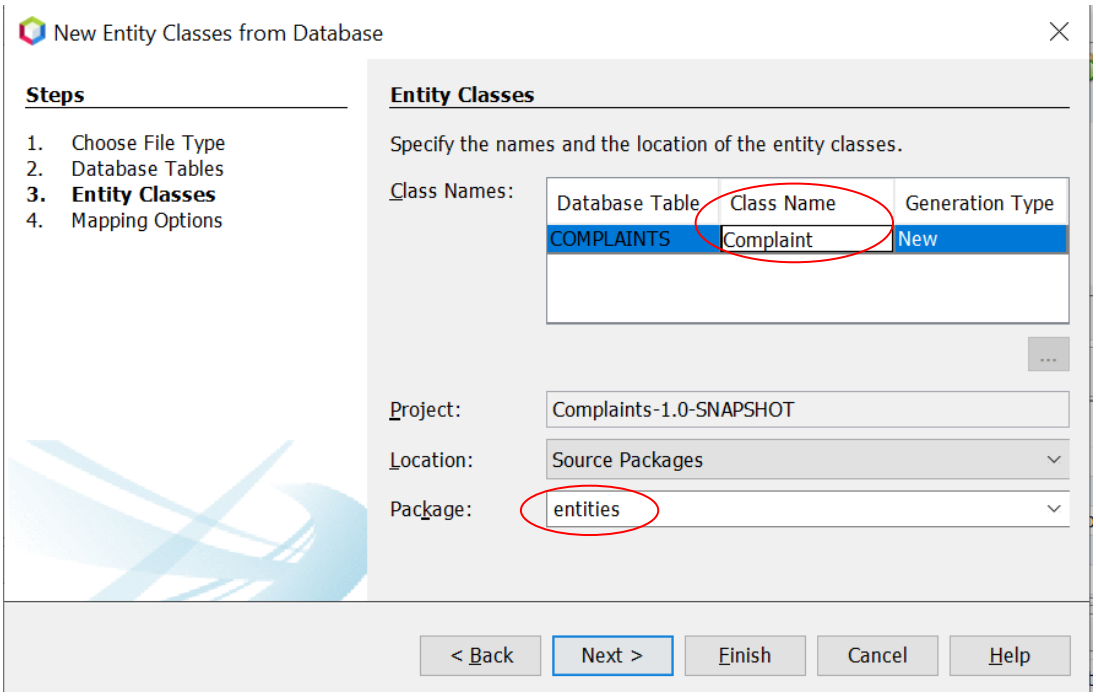
```
DROP TABLE complaints;

CREATE TABLE complaints
 (id INT NOT NULL GENERATED ALWAYS AS IDENTITY
  CONSTRAINT complaints_pk PRIMARY KEY,
  complaint_date DATE,
  complaint_text VARCHAR(60) NOT NULL,
  author VARCHAR(60) NOT NULL,
  status VARCHAR(6) NOT NULL DEFAULT 'open' ,
  CHECK(status IN ('open', 'closed')));

INSERT INTO complaints (complaint_date, complaint_text,
author)
VALUES (CURRENT_DATE, 'Please check TV in room 242',
'Jim Brown');

INSERT INTO complaints (complaint_date, complaint_text,
author)
VALUES (CURRENT_DATE, 'Repair fridge in room 311',
'Arthur McCoy');

INSERT INTO complaints (complaint_date, complaint_text,
author)
VALUES (CURRENT_DATE, 'Remove the blood stains on the
wall in room 242', 'Jim Brown');

INSERT INTO complaints (complaint_date, complaint_text,
author, status)
VALUES (CURRENT_DATE, 'Fix air conditioning in room
242', 'Johny Bravo', 'closed');
```

    c. Upewnij się odpowiednim zapytaniem, że faktycznie zostały dodane cztery wiersze do tabeli COMPLAINTS.

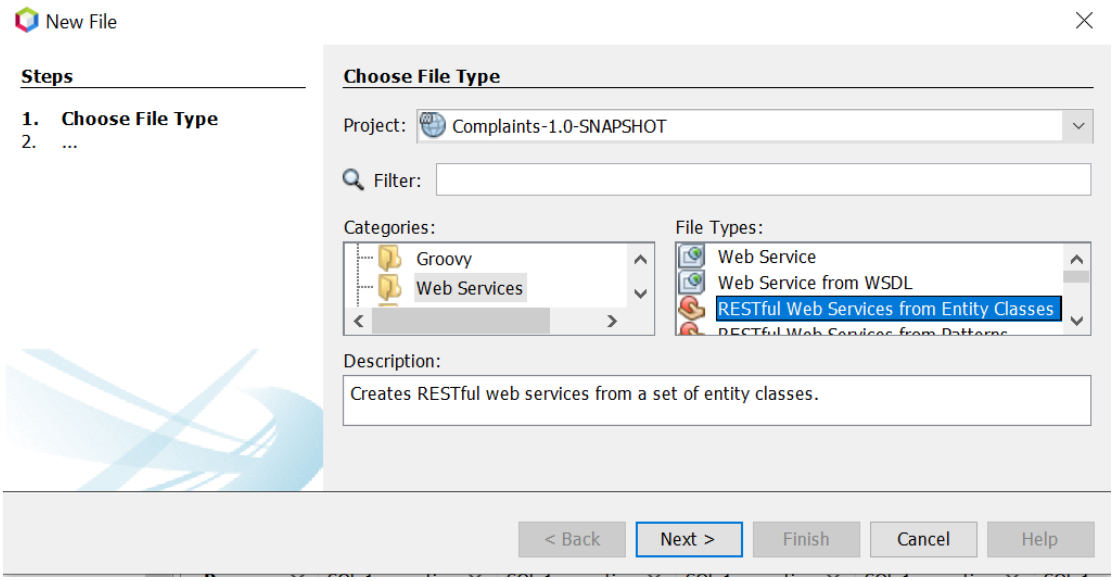
5. W utworzonym projekcie uruchom kreator Entity Classes from Database z kategorii Persistence.
    a. Zleć utworzenie klasy encji odpowiadającej tabeli COMPLAINTS z bazy danych dostępnej poprzez źródło jdbc/sample.



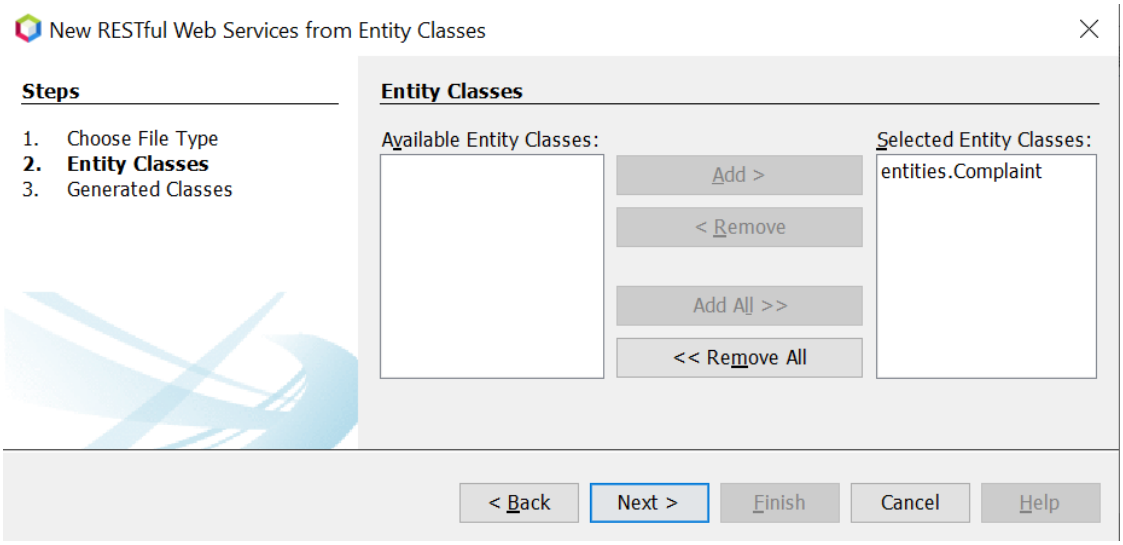
    b. W kolejnym kroku kreatora, dotyczącym tworzonych klas encji, zmień nazwę generowanej klasy na liczbę pojedynczą („Complaint”) i ustaw pakiet na „entities”. Pozostałe opcje pozostaw domyślne.



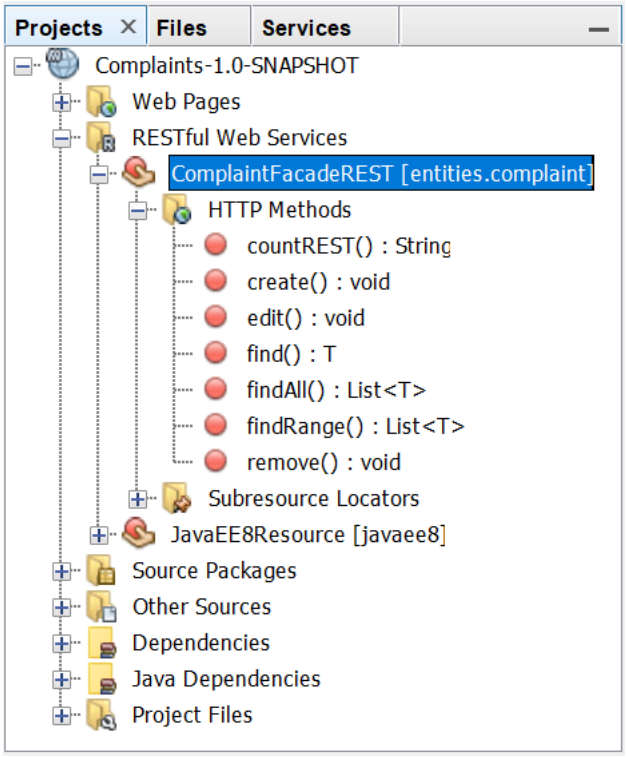
6. Otwórz nowo utworzoną klasę `Complaint`. Zweryfikuj:
    a. Pochodzenie i przeznaczenie poszczególnych adnotacji (JPA, Bean Validation)
    b. Które ograniczenia z bazy danych są reprezentowane w encji przez odpowiadające im ograniczenia Bean Validation?
        - Długości łańcuchów znaków?
        - Ograniczenie na wartości kolumny „STATUS”?
7. Uruchom w projekcie kreator RESTful Web Services from Entity Classes z kategorii Web Services.



8. W kreatorze wybierz klasę encji wygenerowaną w poprzednim kroku ćwiczenia. W ostatnim kroku kreatora pozostaw opcje domyślne.



9. Obejrzyj w strukturze projektu węzeł reprezentujący wygenerowaną usługę REST zwracając uwagę na dostępne operacje. Przejrzyj kod wygenerowanej klasy implementującej usługę zwracając uwagę na adnotacje JAX-RS.



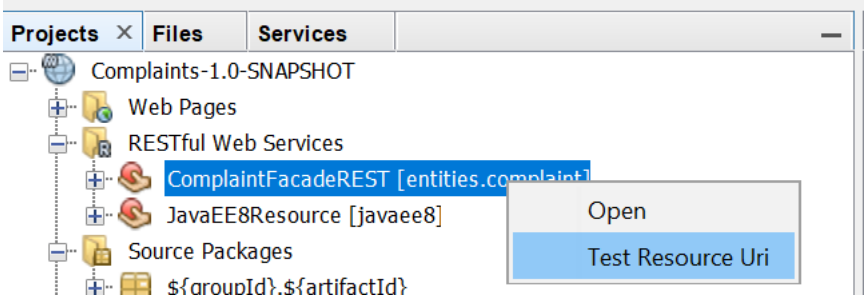
10. Odszukaj w projekcie plik `persistence.xml`. Otwórz go, a następnie przejdź do edycji jego źródła przełączając edytor na zakładkę Source.

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

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

12. Uruchom aplikację.

13. Przetestuj usługę wybierając z menu kontekstowego dla niej opcję Test Resource Uri.



14. Zmodyfikuj adres w przeglądarce dodając do niego człon identyfikujący konkretną skargę np. „/1”.

15. Odszukaj w kodzie źródłowym projektu adnotacje odpowiedzialne za elementy ścieżki prowadzącej do zasobu: „resources” i „entities.complaint”.
16. Poprzez modyfikację odpowiedniej adnotacji spraw by zasób Complaints był dostępny pod adresem http://localhost:8080/Complaints/resources/complaints.
17. Dodaj możliwość sprawdzenia statusu skargi poprzez adres URI.
    a. Otwórz klasę fasady `ComplaintFacadeREST`.
    b. Dodaj metodę `checkStatus` o następującej treści:

```
public String checkStatus(Integer id) {
        return super.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 odczyt statusu dla kilku skarg.
18. 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 te same żądania GET, które sprawdziliśmy w przeglądarce:
    a. Pobranie wszystkich skarg
    b. Pobranie skargi o podanym identyfikatorze
    c. Pobranie statusu skargi o podanym identyfikatorze (w tym przypadku pamiętaj o zmianie nagłówka „Accept” w żądaniu na „text/plain”)
19. 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": "Marvin Doherty",
    "complaintDate": "2020-04-24T11:45:00+02:00",
    "complaintText": "Please fix a tap in room 234",
    "status": "open"
}
```

    e. Zwróć uwagę na kod statusu odpowiedzi HTTP. Co on oznacza?

20. Pobierz z poziomu Postmana wszystkie dostępne skargi i zapamiętaj jaki identyfikator został nadany dodanej przed chwilą skardze.
21. Z poziomu Postmana zaktualizuj dodaną ostatnio skargę, zmieniając w jej treści numer pokoju na 432 i status na „closed”. W tym celu wyślij żądanie metodą PUT wykorzystując sprawdzony przed chwilą identyfikator tej skargi, podając jako treść żądania odpowiedni JSON. Sprawdź żądaniem GET czy modyfikacja została zrealizowana.
22. Dodaj obsługę filtrowania skarg według statusu:
    a. W klasie `ComplaintFacadeREST` do metody `findAll` dodaj parametr typu String o nazwie `status`.
    b. 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.
    c. Zwróć uwagę, że dla parametrów typu query nie trzeba zmieniać ścieżki związanej z metodą klasy
    d. Po dodaniu parametru do metody, przestała ona nadpisywać metodę `findAll` z klasy bazowej – usuń więc adnotację `@Override`
23. Dodaj do metody `findAll` kod, który:
    a. zwróci dotychczasowy rezultat, jeżeli parametr `status` jest pusty (`null`).
    b. 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
  ...
```

24. Uruchom usługę i przetestuj Postmanem działanie filtrowania wg statusu dla zasobu `complaints`.
25. Przetestuj filtrowanie skarg wg statusu bezpośrednio wprowadzając odpowiedni adres w pasku adresu przeglądarki (bez pomocy Postmana).

Ćwiczenie 2

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**, wyczyść pole z nazwą pakietu i kliknij przycisk Finish.

2. Utwórz w utworzonym przed chwilą projekcie klasę `app.Main`.
3. Dodaj w pliku `pom.xml` projektu klienckiego bibliotekę Jersey (implementację JAX-RS):

```
    <dependencies>
        <dependency>
            <groupId>org.glassfish.jersey.core</groupId>
            <artifactId>jersey-client</artifactId>
            <version>2.25.1</version>
        </dependency>
        <dependency>
            <groupId>javax.activation</groupId>
            <artifactId>activation</artifactId>
            <version>1.1.1</version>
        </dependency>
    </dependencies>
```

4. Utwórz w klasie `app.Main` metodę `main()` z poniższym kodem:

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

System.out.println("Count: " + count);

client.close();
```

   Zaimportuj wykorzystywane klasy/interfejsy z pakietów javax.ws.rs.client i javax.ws.rs.core.
5. Uruchom aplikację klienta.
6. Samodzielnie (możesz wzorować się na przykładach np. z Java EE Tutorial) dodaj w metodzie `main()` klienta następujące operacje i przetestuj ich działanie. (Jako format wymiany danych wykorzystaj JSON.)
    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.