

# Egzamin SOA 2020

Marek Matys

# Konfiguracja serwera Wildfly

Nietypowo, musiałem tutaj odpalać serwer komendą:

*start standalone.bat -c standalone-full.xml,*

ponieważ standardowa wersja standalone.xml nie obsługuje JMS.

W związku z tym musiałem ustawić Datasource i JDBC Driver drugi raz dla „rozszerzonej” wersji serwera

# Konfiguracja sterownika bazy danych

Ze względu na niesłynny bug „newValue is null” najpierw musiałem zainstalować nowszą wersję konsoli Hal management console (wersję 3.2.4 w miejsce domyślnej 3.2.1).

Link, pod którym znajduje się dokładny opis jak to zrobić ->

<https://stackoverflow.com/questions/59076182/wildfly-18-0-1-jdbc-drivers-internal-error-newvalue-is-null>

Następnie dodałem sterownik bazy danych za pomocą konsoli – wklejam w kolejnym slajdzie opis jak to zrobić z tutoriala do laboratoriów nr 5 udostępniony przez dr. Rogusa:

# Konfiguracja sterownika bazy danych

## 2. Podejście przy użyciu CLI

Uruchmiamy CLI -> **jboss-cli.bat**

Łączymy się z serwerem aplikacyjnym i rozpoczynamy zabawę:

Potrzebujemy stworzyć nowy moduł do obsługi Postgres. Możemy to zrobić bezpośrednio z linii komend wykonując następującą komendę

```
module add --name=org.postgres --resources=/tmp/postgresql-42.2.5.jar --  
dependencies=javax.api,javax.transaction.api
```

gdzie tmp/postgres..... miejsce w katalogu bin gdzie umieściłem ściągnięty konektor jdbc.

Kolejnym krokiem jest zarejestrowanie stworzonego modułu z poziomu konsoli.

```
/subsystem=datasources/jdbc-driver=postgres:add(driver-name=postgres,driver-module-  
name=org.postgres,driver-class-name=org.postgresql.Driver)
```

# Konfiguracja źródła danych

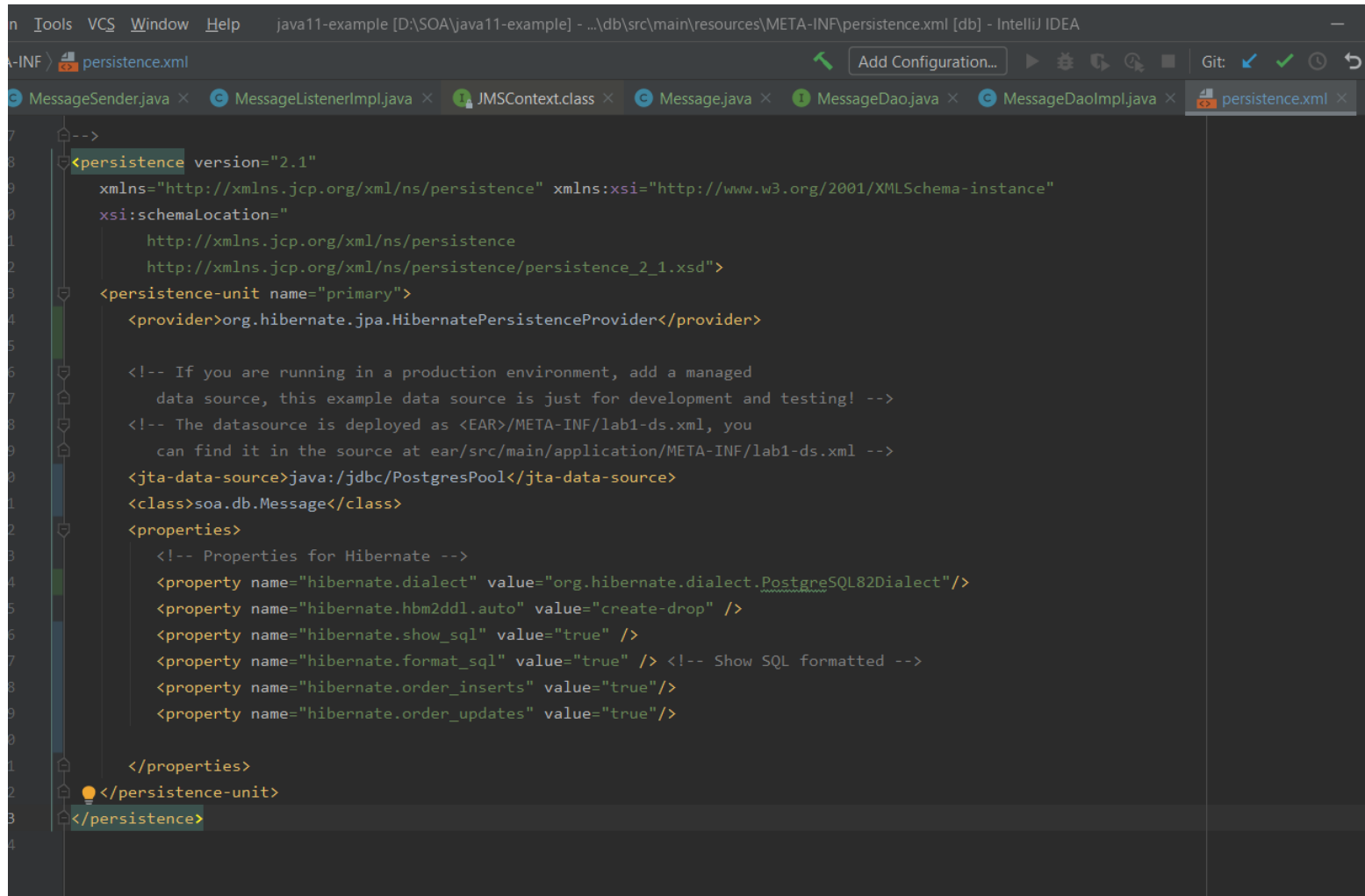
Następnie trzeba skonfigurować nasze DataSource. Oczywiście w miejsce testDB wpisałem nazwę bazy: „marek”, oraz w polach username i hasło: „marek” i „marek”.

Kolejnym krokiem jest stworzenie źródła danych na naszym serwerze. Nie jest to czynność zbyt skomplikowana, aczkolwiek należy podać kilka parametrów. W końcu musimy mieć jakąś bazę danych, do której źródło możemy skonfigurować : >

```
[standalone@localhost:9990 /] data-source add --jndi-name=java:/PostgresDS --connection-url=jdbc:postgresql://localhost:5432/testDB --driver-name=postgres --username=postgres --password=postgres --name=PostgresPool  
[standalone@localhost:9990 /] _
```

Teraz wystarczy tylko za pomocą Webowej Consoli wykonać TestConnection połączenia z bazą.

# Konfiguracja źródła danych (persistence.xml)



The screenshot shows the IntelliJ IDEA IDE with the `persistence.xml` file open. The file is part of a project named `java11-example`. The configuration is for a JPA persistence unit named `primary`, using the `org.hibernate.jpa.HibernatePersistenceProvider` as the provider. It includes a managed data source named `java:/jdbc/PostgresPool` and a class `soa.db.Message`. The properties section is configured for Hibernate with the following settings:

```
<!-- Properties for Hibernate -->
<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQL82Dialect"/>
<property name="hibernate.hbm2ddl.auto" value="create-drop" />
<property name="hibernate.show_sql" value="true" />
<property name="hibernate.format_sql" value="true" /> <!-- Show SQL formatted -->
<property name="hibernate.order_inserts" value="true"/>
<property name="hibernate.order_updates" value="true"/>
```

The IDE interface shows several tabs open: `MessageSender.java`, `MessageListenerImpl.java`, `JMSContext.class`, `Message.java`, `MessageDao.java`, `MessageDaoImpl.java`, and `persistence.xml`. The `persistence.xml` tab is currently active.


# Konfiguracja bazy danych

W postgresql stworzyłem bazę danych „marek”. Utworzyłem użytkownika „marek” z hasłem „marek” i przyznałem mu wszystkie *privileges* dla stworzonej bazy.

Nie trzeba tworzyć ręcznie tabeli, przy deployowaniu jest automatycznie tworzona nowa tabela o nazwie przekazanej jako parametr *name* w adnotacji *@Entity* w pliku *Message.java* z pakietu *soa.db*.

# Konfiguracja autoryzacji

Konfiguracja autoryzacji była na slajdach w udostępnionej dla nas prezentacji nt. SOAP, stosując się do zamieszczonego tam tutorialu krok w krok utworzyłem usera „marek” i grupę „GrupaTestowa”:



## Autoryzacja

Konfigurację można przeprowadzić za pomocą narzędzia `$JBOSS_HOME/bin/jboss-cli.sh`

Jest opcja zrobić to ręcznie wprowadzając zmiany w pliku `%JBOSS_HOME%/configuration/standalone.xml`, gdzie pomiędzy znacznik `<security-domains>` doklejamy kolejny security-domain:

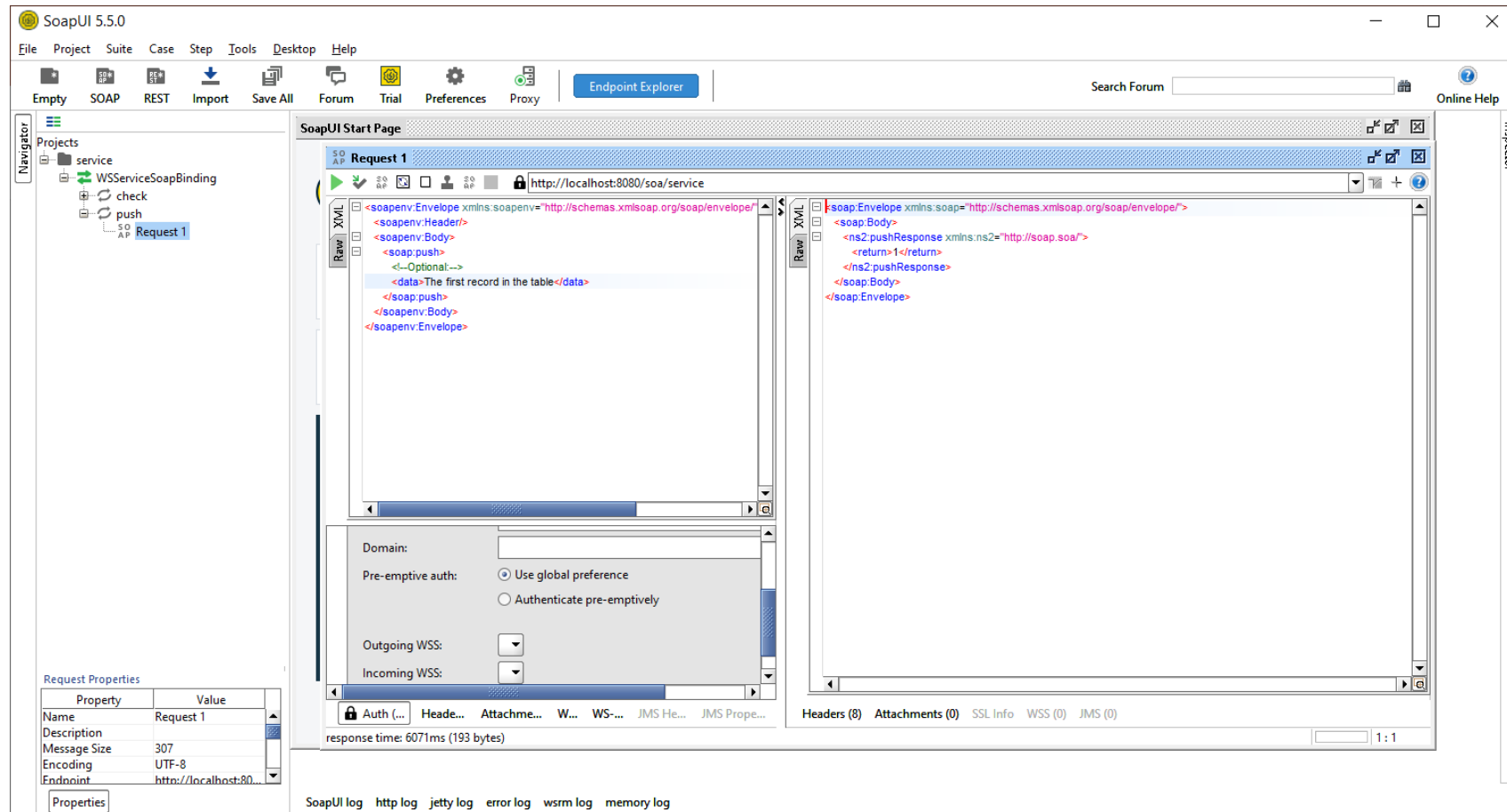
```
<security-domain name="Test-security-domain" cache-type="default">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="usersProperties" value="users.properties"/>
      <module-option name="rolesProperties" value="roles.properties"/>
      <module-option name="unauthenticatedIdentity" value="nobody"/>
    </login-module>
  </authentication>
</security-domain>
```



# Konfiguracja kolejki JMS

Zastosowana kolejka: „java:/jms/queue/DLQ”, czyli domyślna

# Prezentacja działania



Używając narzędzia SoapUI wysyłam request na serwer. W odpowiedzi otrzymuję indeks dodanego rekordu i rekord zostaje dodany do tabeli

# Prezentacja działania

```
C:\WINDOWS\system32\cmd.exe - psql
(1 row)

marek=# \q

D:\wildfly-18.0.1.Final\bin>psql
Password for user marek:
psql (12.2)
WARNING: Console code page (852) differs from Windows code page (1250)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

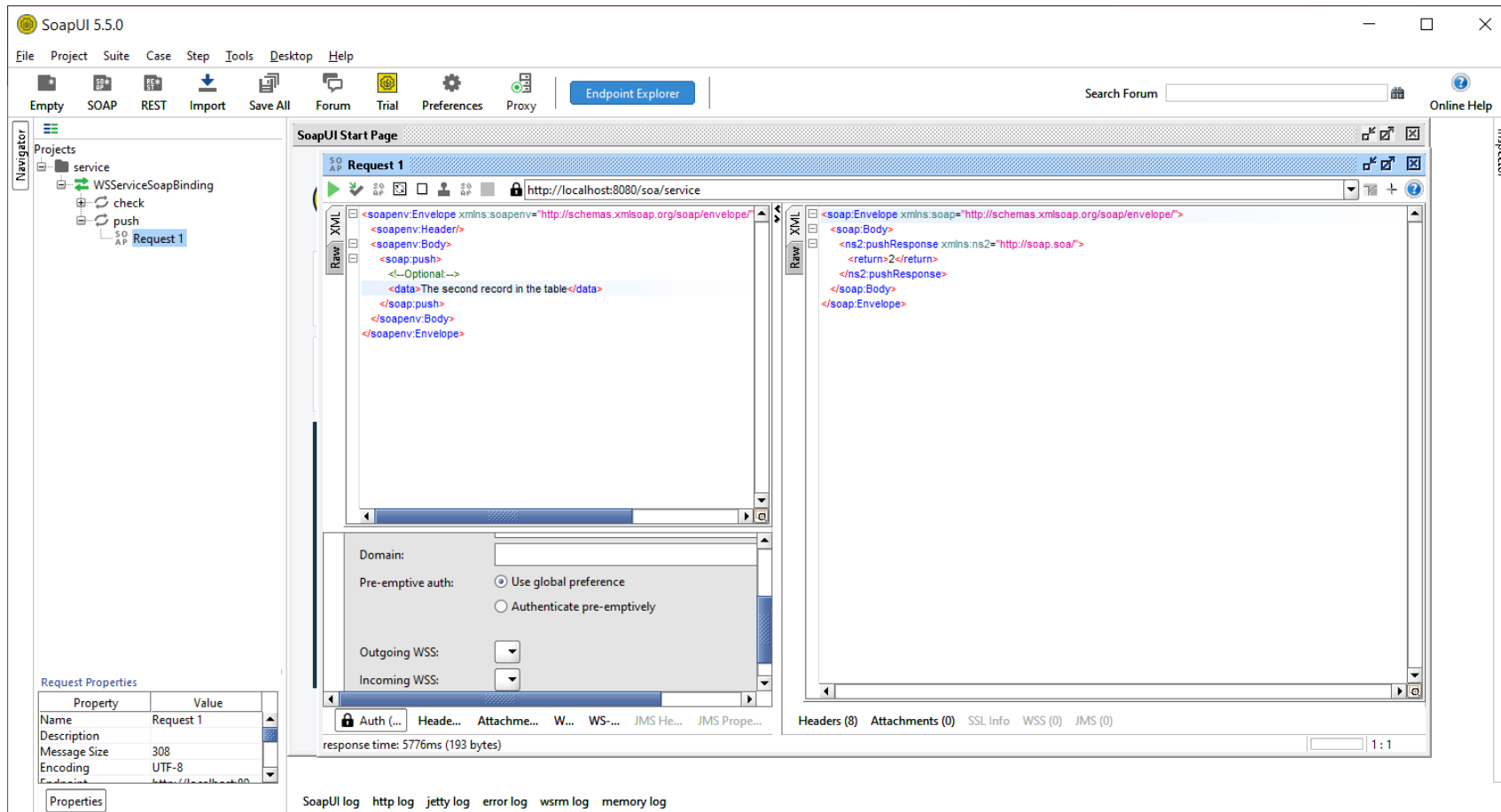
marek=# \dt
      List of relations
 Schema |   Name   | Type  | Owner
-----+-----+-----+-----
 public | books    | table | marek
 public | messages | table | marek
(2 rows)

marek=# select * from messages;
 id |          data          |      time_push      |      time_receive
-----+-----+-----+-----
  1 | The first record in the table | Tue Sep 22 23:52:53 CEST 2020 | Tue Sep 22 23:52:59 CEST 2020
(1 row)

marek=#
```

Sprawdziłem w  
postgresie, czy  
rekordy  
prawidłowo  
dodają się do  
tabeli

# Prezentacja działania



Operacja push po krótkim okresie czasu (wywołanie metody action, czyli 6 do 7 sekund) dodaje wiersz.

# Prezentacja działania

```
C:\WINDOWS\system32\cmd.exe - psql
WARNING: Console code page (852) differs from Windows code page (1250)
        8-bit characters might not work correctly. See psql reference
        page "Notes for Windows users" for details.
Type "help" for help.

marek=# \dt
          List of relations
 Schema |   Name   | Type  | Owner
-----+-----+-----+-----
 public | books    | table | marek
 public | messages | table | marek
(2 rows)

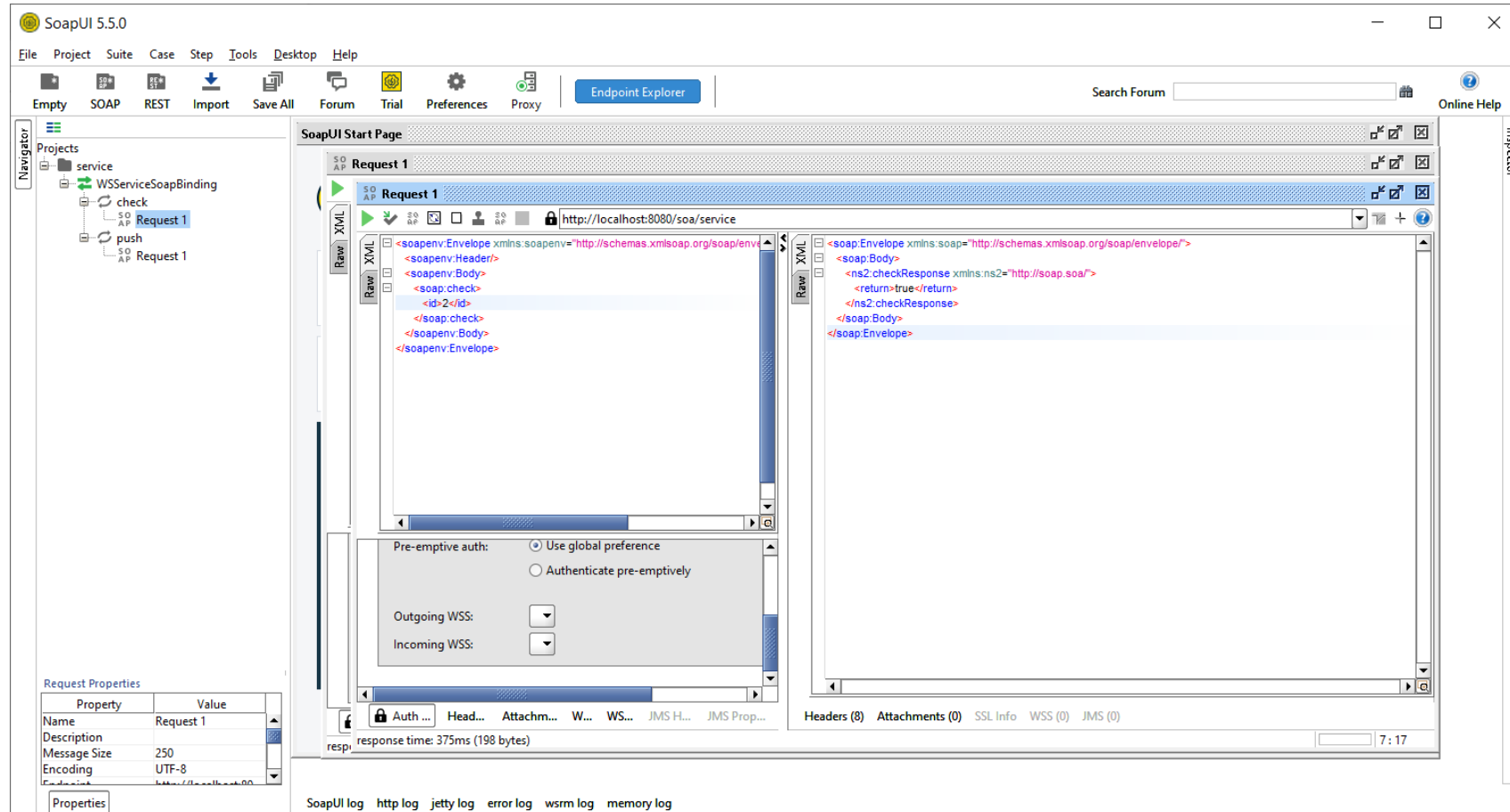
marek=# select * from messages;
 id |          data          |      time_push      |      time_receive
-----+-----+-----+-----
  1 | The first record in the table | Tue Sep 22 23:52:53 CEST 2020 | Tue Sep 22 23:52:59 CEST 2020
(1 row)

marek=# select * from messages;
 id |          data          |      time_push      |      time_receive
-----+-----+-----+-----
  1 | The first record in the table | Tue Sep 22 23:52:53 CEST 2020 | Tue Sep 22 23:52:59 CEST 2020
  2 | The second record in the table | Tue Sep 22 23:58:34 CEST 2020 | Tue Sep 22 23:58:39 CEST 2020
(2 rows)

marek=#
```

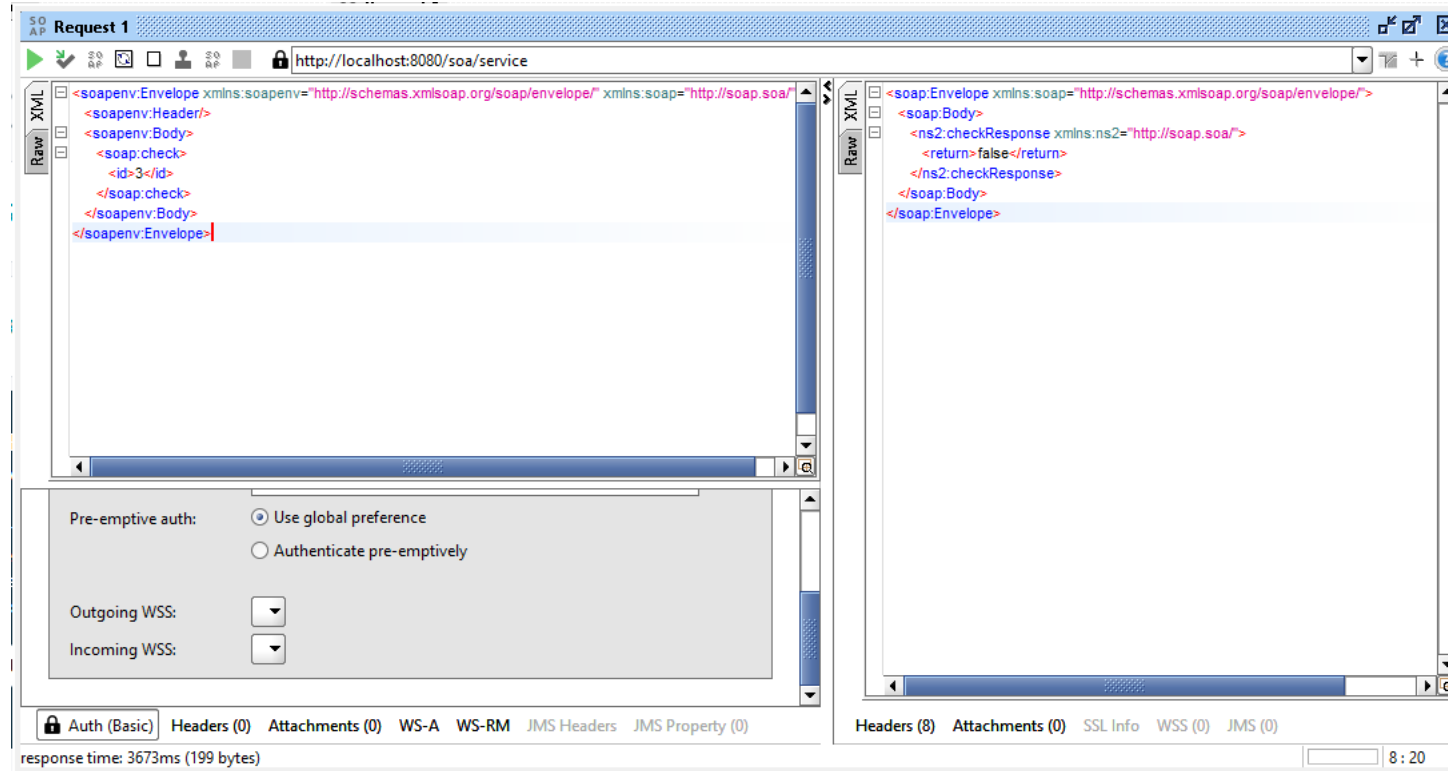
Metoda ta nie wymaga autoryzacji, co za tym idzie, dowolny użytkownik może ją wykorzystać. A to wszystko dzięki adnotacji @PermitAll

# Prezentacja działania



Operacja check wymaga autoryzacji. Zalogowałem się wcześniej utworzoną nazwą użytkownika i hasłem

# Prezentacja działania



Gdy zostanie wywołana metoda push, a nie zostanie zakończona wewnętrzna operacja action, to wywołanie metody check zwraca false

Dziękuję za uwagę!