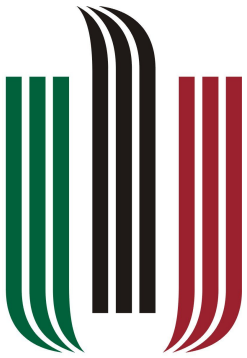
SOA

Projekt zaliczeniowy



AGH

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie
Informatyka 2017

Mateusz Nowak

Wstęp

Aplikacja została podzielona na 4 moduły:

- Parking area service (moduł główny)
- Parking meter mock
- Parking place mock
- Mobile device notifier mock

Do zarządzania wszystkimi zależnościami, modułami oraz do budowania aplikacji wykorzystany został maven.
Poniżej zostanie opisany każdy z nich.

Parking area service (moduł główny)

Moduł główny odpowiada za zasadniczą logikę działania całego systemu. Wystawia restowe i soapowe api, integruje system z poszczególnymi modułami (parkometr, czujnik miejsca parkingowego i system powiadomień), zapewnia połączenie z bazą danych i operacje na niej, odpowiada za wykrywanie nieprawidłowych stanów (np. zajęte miejsce parkingowe bez biletu lub z przedawnionym biletem), udostępnia dashboard dla pracowników, na którym znajduje się aktualny stan systemu oraz odpowiada za autentykację i autoryzację dostępu do niego.
API RESTowe zostało wykonane przy pomocy biblioteki RESTEasy. Wszystkie endpointy rozpoczynają się od prefixu "/rest" i pozwalają na dokonanie podstawowych zapytań. Nie zostały dopuszczone jednak wszystkie metody ze względu na utrzymanie spójności danych (chodzi o to, że nie przewiduje możliwości usunięcia czy modyfikacji strefy, ulicy itd.).
API SOAPowe umożliwia zajęcie i zwolnienie miejsca parkingowego (na potrzeby parking place mocka) oraz pobranie aktualnego stanu stref parkingowych.
Mapowanie obiektowo-relacyjne odbywa się przy pomocy frameworku Hiberante. Konfiguracja modelu danych wykonana została w całości z wykorzystaniem adnotacji. Wykorzystana została biblioteka Lombok, która dzięki odpowiednim adnotacjom generuje nam gettery i settery oraz metody equals i hashCode dla danej klasy.
Wykorzystywana baza danych to PostgreSQL w wersji 9.5. Dodany został skrypt import.sql, który wykonywany jest w trakcie uruchamiania aplikacji i wrzuca on przykładowe dane (użytkowników, strefy, ulice, parkometry i miejsca parkingowe) do bazy.
Wykrywanie nieprawidłowego stanu miejsc parkingowych odbywa się asynchronicznie w klasie IllegalStateDetector z wykorzystaniem Java Concurrency API. W klasie trzymana jest referencja do biletu, którego ważność jest najkrótsza. Ustawione jest wywołanie metody na datę wygaśnięcia biletu, która wysyła powiadomienie do kolejki o nieprawidłowym stanie a następnie szuka kolejnego biletu, którego ważność kończy się najszybciej. W przypadku gdy nowy bilet wygasa szybciej niż ten, do którego referencję przechowujemy w klasie wykonywana jest odpowiednia podmiana.
Dashboard wykonany został przy pomocy technologii JSF wraz z dodatkowymi bibliotekami Bootsfaces i Primefaces. Biblioteka Bootsfaces umożliwia nam wykorzystanie Bootstrapa do

prezentacji danych. Z kolei biblioteka Primefaces wykorzystywana jest do odświeżania widoku co 30 sekund. Główny widok zawiera tabelę z miejscami parkingowymi i ich aktualnym stanem, widok raportów dla administratora strefy zawierający statystyki oraz możliwość pobrania danych w formacie PDF (do generowania pliku PDF wykorzystywana jest biblioteka iText) oraz widok zmiany hasła, które hashowane jest przy pomocy algorytmu SHA-256.
Autentykacja odbywa się poprzez dodanie obiektu zalogowanego użytkownika do sesji co umożliwia filtrowanie danych widocznych dla użytkownika oraz schowanie/pokazanie odpowiednich kontrolek na pasku nawigacyjnym. Trzymana jest mapa zalogowanych użytkowników w ich sesje, dzięki czemu można zablokować możliwość zalogowania się do aplikacji więcej niż jeden raz.
Zaimplementowany został filtr umożliwiający dostęp do aplikacji jedynie przy pomocy przeglądarki Chrome.

Parking meter mock

Moduł służący do zasymulowania działania parkometru. Składa się z formularza, do którego podaje się id parkometru oraz czas na jaki chce się kupić bilet. Następnie wykonywany jest zapytanie post na odpowiedni endpoint api restowego gdzie bilet zapisywany jest w bazie danych i przypisywany do odpowiedniego miejsca parkingowego.

Parking place mock

Moduł służący do zasymulowania działania czujnika zajętości miejsca parkingowego. Składa się z formularza, w którym podaje się id miejsca parkingowego i wybiera akcję (take/zajmij, leave/opuść) poprzez naciśnięcie odpowiedniego przycisku. Następnie przy pomocy soapa wykonywana jest metoda po stronie modułu głównego, która uaktualnia stan o zajętości miejsc parkingowych.

Mobile device notifier mock

Moduł służący do zasymulowania działania systemu alertów dla kontrolerów stref parkingowych. Zaimplementowany został jako Message Driven Bean wyświetlający wiadomości otrzymywane z kolejki “notification-queue”, do której dane wysyłane są w wyniku odpowiednich akcji z modułu głównego takich jak zajęcie/opuszczenie miejsca parkingowego, zakup i przypisanie biletu oraz przedawnienie biletu już przypisanego do konkretnego miejsca.

Konfiguracja

Do uruchomienia aplikacji potrzebujemy zbudować poszczególne moduły przy pomocy mavena. Potrzebna będzie konfiguracja datasource i samej bazy danych dla projektu. W PostgreSQL należy założyć bazę danych “projekt”.
Kontener aplikacyjny wildfly należy odpalać z opcją “-c standalone-full.xml”. Należy dodać do modułów w wildfly moduł PostgreSQL wraz z odpowiednim driverem.

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.postgres">
     <resources>
          <resource-root path="postgresql-9.4.1212.jar"/>
     </resources>
     <dependencies>
          <module name="javax.api"/>
          <module name="javax.transaction.api"/>
     </dependencies>
</module>
```

Następnie należy dodać ten moduł do pliku konfiguracyjnego standalone-full.xml.

```
<datasources>
     …
     <drivers>
          …
          <driver name="postgresql" module="org.postgres"/>
          …
     </drivers>
     …
</datasources>
```

Kolejno z poziomu webowej konsoli administracyjnej wildfly’a trzeba dodać datasource. Wybieramy datasource dla PostgreSQL, wpisujemy nazwę (projekt-ds) i JNDI (java:jboss/datasources/projekt-ds), wybieramy wykryty sterownik dla postgresa, wpisujemy adres bazy (w moim przypadku jdbc:postgresql://localhost:5432/projekt), użytkownika i hasło (w moim przypadku admin/admin) i zatwierdzamy.
Następnie trzeba dodać kolejkę “notification-queue” wykorzystywaną przez aplikację. W tym celu również można wykorzystać webową konsolę. Wybieramy odpowiedni kreator, wpisujemy nazwę (notification-queue) i JNDI (java:/jms/queue/notification-queue).
Taka konfiguracja powinna wystarczyć do uruchomienia aplikacji.